Cet article est rédigé par des élèves. Il peut comporter des oublis ou des imperfections, autant que possible signalés par nos relecteurs dans les notes d'édition.

Les mathématiques du Tetris

Année 2025-2026

Nargès DINANE, Mehdi EL GARF, Rhode KHAROMON, élèves de terminale et de première

Établissement : Lycée Jean Zay, Aulnay-Sous-Bois
Enseignante : Majuri MANOHARAN
Chercheurs : Flavien BREUVART, Axel FOUDA (étudiant Polytechnicien)

Table des matières

1. Introduction

Tetris est un jeu populaire qui émerge dans les années 80. Les règles du jeu paraissent simples, pourtant au sein de ce jeu se cachent d'énormes propriétés mathématiques. Des questions peuvent se poser sur les polyominos étudiés.

Nous avons répondu à plusieurs questions durant ces recherches :

— Peut-on remplir un échiquier à l'aide de 15 tétraminos en T et un tétraminos carré ?

— Peut-on paver un rectangle avec un seul exemplaire de chaque tétraminos ?

— Combien existe-t-il de pentaminos, heptaminos, hexaminos, octominos, n-minos ?

2. Conditions initiales

Nos sujets de recherche s'étendent sur les polyominos : des figures composées de n carrés avec $n \in \mathbb{N}$. De plus, on les considère connexes (nous pouvons aller d'un carré à n'importe quel autre sans sortir de la figure).

Ce sont les propriétés primordiales pour mener nos recherches.

3. Pavage

3.1. Pavage d'échiquier

On considère un échiquier classique noir et blanc de dimension 8×8. On se pose la question suivante : peut-on remplir un échiquier à l'aide de 15 tétraminos en T et un tétramino carré ?

On note X_1 les tétraminos en T et X_2 les tétraminos carrés. On a : $X_1 = (1;3)$, $X_2 = (3;1)$ avec (cases blanches ; cases noires).

On pose $(k_1, k_2) \in [\![1;15]\!]^2$.

$(2;2)$ correspond au tetraminos carré.

$(2;2) + k_1X_1 + k_2X_2$ une expression qui donne le nombres de cases blanches et noires.

Démontrons qu'il n'existe pas de $(k_1, k_2) \in [\![1;15]\!]^2$ tels que $(2;2) + k_1X_1 + k_2X_2 = (32;32)$. Nous allons montrer que nous n'avons pas la condition nécessaire pour paver l'échiquier.

De plus, $k_1 + k_2 = 15$.

On remplace et on développe.

$(2;2) + k_1(1;3) + k_2(3;1) = (32;32)$.

$(2 + k_1 + 3k_2; 2 + 3k_1 + k_2) = (32;32)$.

On résout le système suivant : $(S)\begin{cases} 2 + k_1 + 3k_2 &= 32 \\ 2 + 3k_1 + k_2 &= 32 \\ k_1 = 15 - k_2 & \end{cases}$

$(S) \Leftrightarrow \begin{cases} 2 + 15 - k_2 + 3k_2 &= 32 \\ 2 + 3(15 - k_2) + k_2 &= 32 \\ k_1 = 15 - k_2 & \end{cases} \Leftrightarrow \begin{cases} k_2 &= \dfrac{15}{2} \\ k_1 &= \dfrac{15}{2} \end{cases}$

Or, $(k_1, k_2) \notin [\![1;15]\!]^2$. Donc la condition nécessaire n'est pas vérifiée. Donc on ne peut pas paver l'échiquier.

3.2. Pavage d'un rectangle

On peut considérer cette fois-ci des rectangles de dimensions quelconques. On se pose la question suivante : Peut-on paver un rectangle avec un seul exemplaire de chaque tétramino ?
En prenant le modèle de l'échiquier, on décompose, par exemple, un rectangle de dimension 7 (28 cases correspondant aux nombres de monominos nécessaires pour construire un exemplaire de chaque tétramino) en 14 cases blanches et 14 cases noires comme ci-dessous :

On note chaque tétramino sous la forme $X \in \mathbb{Z}^2$ avec $(x; y)$ avec $x = N$ et $y = B$:

Tous les tétraminos ayant des bases $(2;2)$ sont négligeables car leurs différences valent 0. Donc le tétraminos en T ($(3;1)$ ou $(1,3)$) n'est pas négligeable.
Si le pavage est possible, alors la somme des bases valent $(14;14)$. Vérifions cela :
$6 \times (2;2) + (3;1) = (12;12) + (3;1) = (15;13) \neq (14;14)$.
Il y a un excès de N et un défaut de B.
De même : $6 \times (2;2) + (1;3) = (12;12) + (1;3) = (13;15) \neq (14;14)$.
Donc peu importe la base de tétraminos en T choisi, il rend le rectangle d'aire 28 impossible à paver malgré totale des 7 tétraminos qui vaut 28.

4. Dénombrement

4.1. Compter les nombres de n-minos

Pour pouvoir compter le nombre de polyominos possibles pour un nombre de carrés donné, un programme Python a été conçu pour que la tâche ne soit plus laborieuse. Pour que le langage Python puisse comprendre le concept de polyomino, il faut utiliser des valeurs numériques, car l'ordinateur travaille avec les nombres plutôt que de tels concepts. Pour cela, on place les polyominos dans un repère, de telle sorte à ce que les cases constituant les polyominos aient des coordonnées entières. Donc, pour l'ordinateur, un polyomino est un ensemble de points à coordonnées entières. En utilisant un tel système de coordonnées pour nos polyominos, l'ordinateur comprend le concept, mais plusieurs précisions superflues ont été ajoutées, qui viennent poser trois problèmes majeurs à la validité d'un polyomino.

Avant de traiter les problèmes, il faut savoir comment le polyomino est construit. On pose une case initiale, de coordonnées $(0,0)$. Puis, en partant de cette case, on ajoute ou retire 1 à la première ou deuxième coordonnée, mais jamais les deux coordonnées en même temps. On obtient une nouvelle case adjacente à celle de départ. A partir de la case initiale et de la nouvelle, on répète le processus, en faisant attention à ne pas retomber sur une case déjà existante en faisant l'opération. Pour que l'ordinateur puisse vérifier si la case n'est pas déjà comptée, on fait une prévention : on effectue l'opération de création de case adjacente sans l'ajouter au polyomino. On effectue cette opération pour toutes les possibilités de cases possibles voisines au polyomino. Si l'un des cas mène à la création d'une case se trouvant déjà dans le polyomino, alors le résultat de l'opération n'est pas ajouté dans le polyomino, et les autres cas sont traités. On répète le processus, autant de fois qu'il faut pour obtenir le nombre de cases souhaité dans le polyomino.

Cependant, trois problèmes apparaissent lors de la conception de tous les polyominos possibles pour un nombre de cases donné.

Le premier est le problème des translations : deux polyominos qu'on peut considérer similaires visuellement, en tant qu'humain, peuvent être considérés comme deux polyominos différents selon l'ordinateur. Cette distinction est due au décalage éventuel de coordonnées des deux polyominos. Pour remédier à cela, on prend les coordonnées de la case la plus en bas à gauche du polyomino, et on les soustrait aux coordonnées de toutes les cases du polyomino. Le polyomino résultant est donc translaté vers l'origine. Lors de la création de tous les polyominos possibles pour un nombre de cases donné, cette translation est appliquée à tous les polyominos pour pouvoir résoudre le problème.

Le deuxième problème est le problème des rotations. Encore une fois, deux polyominos qu'on peut considérer similaires, à des rotations près, peuvent être considérés comme deux polyominos différents selon l'ordinateur. Cette distinction est encore due aux coordonnées des deux polyominos qui diffèrent complètement après une rotation. Pour remédier à cela, on associe à chaque polyomino, un ensemble de rotations de 90°, de 180°, de 270°, et 0°. Par conséquent, si deux polyominos sont les mêmes par rotation, alors devraient partager le même ensemble de rotations. Donc, dans la liste contenant tous les polyominos possibles pour un nombre de cases donné, on n'y mettra plus des polyominos, mais des ensembles de rotations de ces polyominos.

Pour dire si deux polyominos sont similaires, il suffirait donc de comparer leurs ensembles de rotations. Mais un troisième problème survient : il y a aussi des décalages entre les ensembles de rota-

tions au niveau des coordonnées. Pour supprimer totalement le problème de décalage, on compare les «tracés» des polyomino. Le tracé d'un polyomino est un ensemble de coordonnés qui est construit comme suit; tout d'abord, il faut ordonner les cases des polyominos. Pour cela, on crée un ordre qui nous permettra d'attribuer un numéro à chaque case du polyomino. On part de la case en bas à gauche du polyomino, puis on numérote les cases de bas en haut. S'il n'y a plus de cases en allant plus haut, on se dirige vers la droite, et on recommence à numéroter de bas en haut, et ce jusqu'à numéroter toutes les cases du polyomino.

Ensuite, il faut construire le tracé à l'aide de la numérotation. Pour chaque case, en allant dans l'ordre croissant des numéros, on soustrait la coordonnée d'une case à la coordonnée de la case comportant le numéro précédent. Comme la première case (celle en bas à gauche) n'a pas de case la précédant, on ne la compte pas dans le tracé. Les différences de coordonnées sont relevées dans une liste dans le même ordre croissant. Cette liste est donc notre tracé.

Finalement, pour dire si deux polyominos sont similaires, il faut comparer les tracés des polyominos se trouvant dans les ensembles de rotations.

Une fois que la conception de la liste contenant tous les polyominos possibles pour un nombre de cases donné est terminée, avec les solutions appliquées aux problèmes, il faut exclure les doublons. Pour les exclure, on parcourt la liste, et on ajoute les éléments de la liste rencontrés dans une nouvelle liste vide. Avant d'ajouter de nouveaux éléments dans la liste initialement vide, on vérifie si l'élément parcouru n'est pas déjà présent. Si l'élément est déjà présent, on ne l'ajoute pas, et on l'ajoute dans le cas contraire.

Après avoir filtré la liste en excluant les doublons, il suffit de compter les éléments dans la liste de polyominos pour avoir le nombre de polyominos possibles pour un nombre de cases donné.

4.2. Construction d'un programme Python

```python
def combinaisons(n,eval_set:list=[],increment:int=1):
    n -= 1
    if n == 1: return 1
    if increment == 1:
        base = poly([coord(0,0)]),
    else:
        base = eval_set
    new = []
    for i in base:
        for j in i.voisins():
            new.append(poly(i.poly+[j]))
    new = unique(new)
    newincr = increment + 1
    if increment == n:
        return len(new)
    n += 1
    return poly.combinaisons(n,new,newincr)
```

```
class poly:
    def __init__(self,coords:list[coord] = []):
        self.poly: list[coord] = unique(coords)
    def ajouter(self,c):
        if c not in self.poly:
            self.poly.append(c)
            self.poly = sort_order(self.poly,coord.totalordergt)
    def rang(self):
        return len(self.poly)

    def etendre(self,*cases):
        for e in cases:
            self.ajouter(e)
        self.poly = sort_order(self.poly,coord.totalordergt)
    def __repr__(self):
        return f"poly{self.poly}"
    def connexe(self):
        for i in flatten(self.poly):
            if not any([(i+s) in self.poly for s in shift]):
                return False
        return True
    def snap_to_0(self):
        if self.rang() == 1:
            self.poly == [coord(0,0)]
        else:
            P = lsub(self.poly,order(self.poly,coord.totalordergt))
            self.poly = P
```

```
class poly:
    def __init__(self,coords:list[coord] = []):
        self.poly: list[coord] = unique(coords)
    def ajouter(self,c):
        if c not in self.poly:
            self.poly.append(c)
            self.poly = sort_order(self.poly,coord.totalordergt)
    def rang(self):
        return len(self.poly)

    def etendre(self,*cases):
        for e in cases:
            self.ajouter(e)
        self.poly = sort_order(self.poly,coord.totalordergt)
```

4.3. Pentaminos, hexaminos, heptaminos

En exécutant le programme pour 5 cases dans le polyomino, on tombe sur 18 possibilités, ce qui correspond au résultat trouvé à la main. Pour 6 cases, à l'aide du programme, il existe 60 hexaminos différents et pour 7 cases, il existe 196 heptaminos différents.

5. Quelques pistes pour la suite des recherches

Nous avons commencé à réfléchir sur les questions suivantes.
Peut-on paver un rectangle avec un seul exemplaire de chaque tétramino et deux tétraminos en T ?
Idées : si l'on prend deux tétraminos en T , alors on retombe sur $(4;4)$. Ce qui revient à prendre 2 tétraminos quelconques (sauf le T). Ainsi si l'on prend une longueur et une largeur paire alors serait-il possible de paver n'importe quel rectangle ?