Rapport du lab03

Emilie CHEN

Les images sont tous issus du sujet, mais ont subi plus ou moins de modifications pour une meilleure compréhension.

Exercices résolus

Quiz 1 :

Ce quiz consiste à placer un tetromino ‘T’ d’une manière donnée.
L’image 1.1 explique que nous souhaitons déplacer le tetromino de deux cases à gauche et d’une rotation vers la gauche de 90°, comme nous pouvons le voir à l’image 1.1.

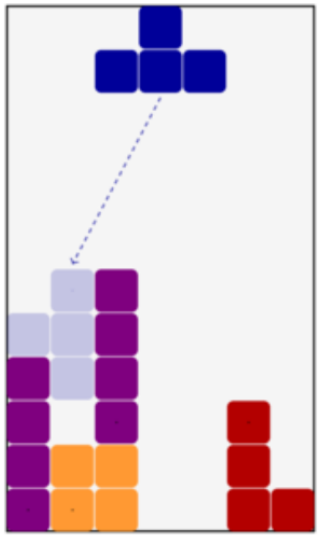


Image 1.1 Schéma indiquant l’emplacement voulu du tetromino

Les méthodes appliquées à un tetromino ou à un board ne modifient pas l’objet en lui-même. Ces méthodes consistent à créer un nouveau objet pour remplacer l’ancien.

Dans le code donné, le board et le tetromino piece ont déjà été implémentés. Il suffit de créer le tetromino à l’emplacement voulu et créer un board le contenant.

Pseudo code :

```
# Créer un tetromino qui est celui obtenu après avoir déplacé piece de deux
cases à gauche et d’une rotation vers la gauche de 90°.

# Créer un tetromino qui est celui obtenu après avoir posé new_piece tout en
bas du board.

# Créer un board qui contient le tetromino new_piece2.
```

Quiz 2 :

Nous voulons à présent implémenter une fonction qui retournerait :
- une liste de grids associée à toutes les possibilités de placement d'un tetromino sur le board
- une liste de tetrominos utilisés pour générer la liste de grids.

Pour mieux appréhender l'exercice, nous pouvons regarder l'image 2.1. En effet, pour le tetromino de type 'I', il est question de l'orienter verticalement et de le déposer sur chaque colonne, puis faire la même chose en l'orientant horizontalement. Ainsi, nous obtenons une liste exhaustive de l'état après dépôt du tetromino.

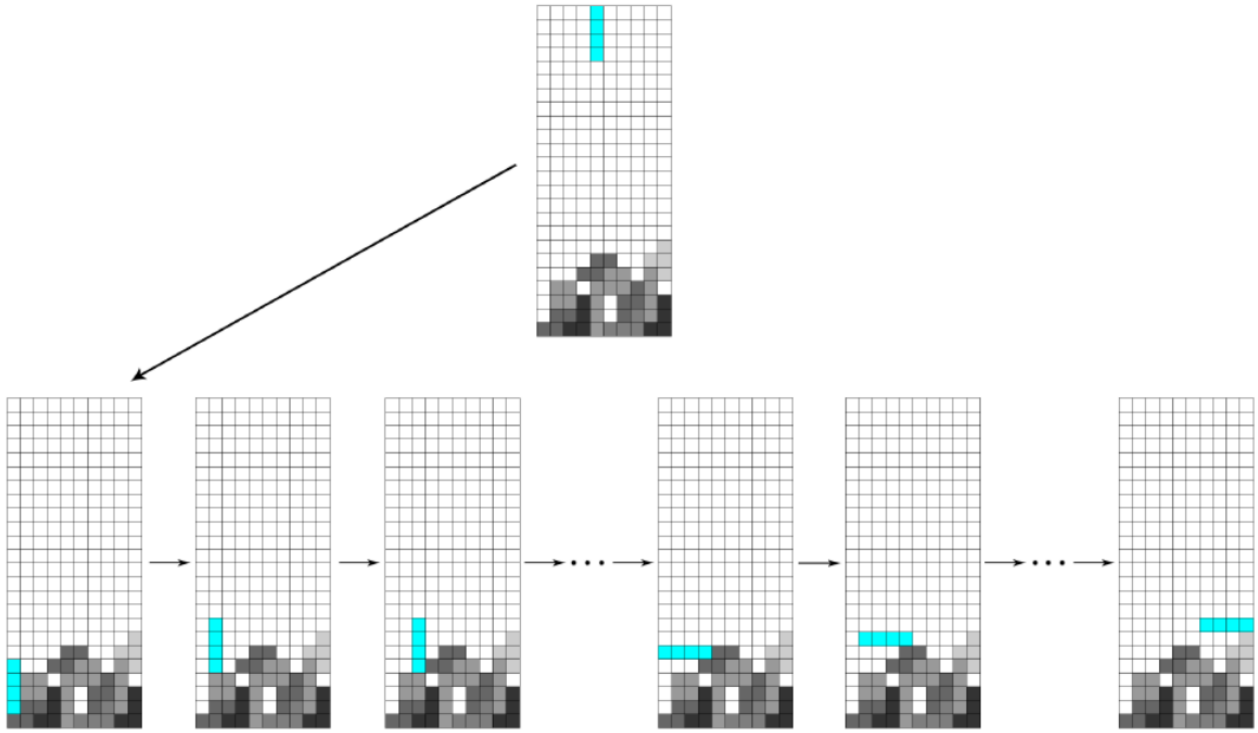


Image 2.1 Liste de toutes les possibilités d'emplacement d'un tetromino

Un tetromino peut prendre différentes formes après une rotation de 90° vers la gauche, une rotation de 90° vers la droite et une rotation de 180°.

Dans le code donné, le nombre d'orientations possibles n de chaque type de tetromino sont listées dans une liste orientation : orientations = {'T': 4, 'J': 4, 'L': 4, 'Z': 2, 'S': 2, 'I': 2, 'O': 1}

En effet, pour avoir toutes les possibilités d'emplacement :
- Pour le tetromino 'O', il n'y a pas de rotation à faire
- Pour les tetrominos 'Z', 'S' et 'I', il n'est pas nécessaire d'effectuer une rotation de 180°
- Pour les tetrominos 'T', 'J' et 'L', il faut effectuer les trois rotations

Pseudo-code :

```
# Pour toutes les orientations
       # Pour toutes les colonnes
            # Créer un tetromino qui est celui obtenu après dépôt
            # Passer à la colonne suivante si le tetromino ne peut pas être posé.
            # Sinon, créer un board qui contient le tetromino posé.
            # Ajouter le board et le tetromino à 'grids' et 'tetros' respectivement
#Retourner 'grids' et 'tetros'
```

Quiz 3 :

Ce quiz consiste à créer une fonction qui liste la hauteur des colonnes.

La solution choisie est d'effectuer une boucle for sur les colonne puis d'utiliser une fonction donnée pour obtenir les indices des éléments non nuls de chaque colonne. La hauteur d'une colonne sera égale à la taille de la colonne soustrait à l'indice du premier élément non nul.
Si une colonne est vide, nous posons simplement que la hauteur de la colonne est nulle. Pour illustrer, nous pouvons nous référer à l'image 3.1.

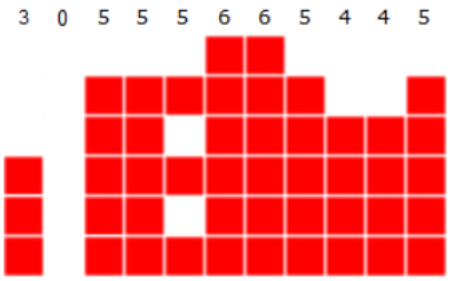


Image 3.1 Schéma indiquant la hauteur des colonnes d'un board

La fonction retourne :

```
    TEST BOARD
   o----------------------o
 0 |           [][]       |
 1 |     [][][][]   []     []|
 2 |     [][]   []   [][][][]|
 3 |[]   [][][][][][][][][][]|
 4 |[]   [][]   [][][][][][]|
 5 |[]   [][][][][][][][][][]|
   o----------------------o

Column height: [3 0 5 5 5 6 6 5 4 4 5]
```

Quiz 4 :

Ce quiz consiste a calculé la hauteur maximal du board en sommant la hauteur des colonnes. Un exemple de calcul a été représenté par l'image 4.1.
En effet, il est intéressant de connaître cette valeur, car l'IA va chercher à le minimiser pour ne pas dépasser le board et perdre.

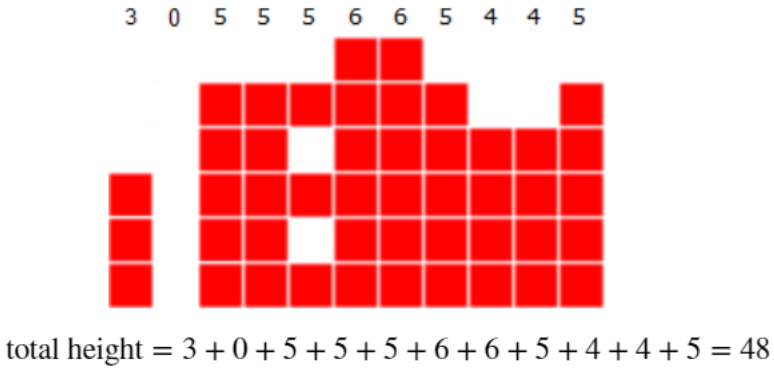


Image 4.1 Schéma indiquant la hauteur totale d'un board

La fonction retourne :

```
    TEST BOARD
   o----------------------o
 0 |            [][]     |
 1 |     [][][][]   []     []|
 2 |     [][]   []   [][][][]|
 3 |[]   [][][][][][][][][]|
 4 |[]   [][]   [][][][][][]|
 5 |[]   [][][][][][][][][]|
   o----------------------o

Total height: 48
```

Quiz 5 :

Une ligne complète est une ligne sans trou. Un exemple a été représenté par l'image 5.1. Le nombre de libre complète est une valeur que cherche à maximiser l'IA pour pouvoir les enlever et avoir plus d'espaces. Nous devons pour ce quiz implémenter une fonction qui compte le nombre de lignes complètes d'un board.

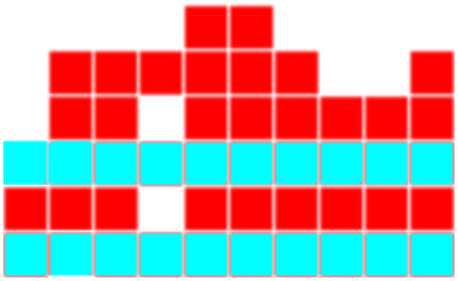


Image 5.1 Schéma indiquant le nombre de lignes complètes d'un board

Pseudo-code :

```
#Initialiser le compteur 'lines' à 0
#Sommer les valeurs du grid sur les lignes
#Pour toutes les lignes
      Incrémenter 'lines' si la somme associée est égale à la taille d'une ligne
#Retourner 'lines'
```

La fonction retourne :

```
    TEST BOARD
   o--------------------o
 0 |         [][]       |
 1 |   [][][][]   []    []|
 2 |   [][]   []   [][][][]|
 3 |[][][][][][][][][][]|
 4 |[][][][]   [][][][][][]|
 5 |[][][][][][][][][][]|
   o--------------------o

Complete lines: 2
```

Quiz 6 :

Ce quiz a pour but d'implémenter une fonction qui compte le nombre de trou dans le board. Un trou est une case vide telle qu'il existe au moins une case remplie au dessus de lui, sur la même colonne. Un exemple a été représenté par l'image 6.1.

Nous allons parcourir chaque case à l'aide de deux boucles for. Pour chaque case, nous allons vérifier s'il s'agit d'un trou.

Un trou est une case vide tel qu'il existe une case remplie au dessus de lui. Autrement dit, une case est un trou si elle est vide et que la hauteur de la colonne est plus grande que sa position à partir du bas.

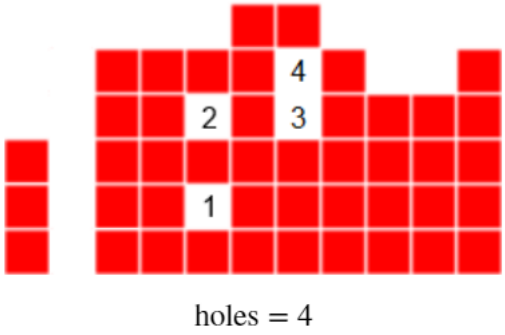


Image 6.1 Schéma indiquant le nombre de trou d'un board

Pseudo-code :

```
#Initialiser le compteur 'holes' à 0.
#Pour toutes les lignes 'i'
      #Pour toutes les colonnes 'j'
            #Si la case (i,j) est vide et qu'une case remplie est au dessus de lui
                  #Incrémenter 'holes'
#Retourner 'holes'
```

La fonction retourne :

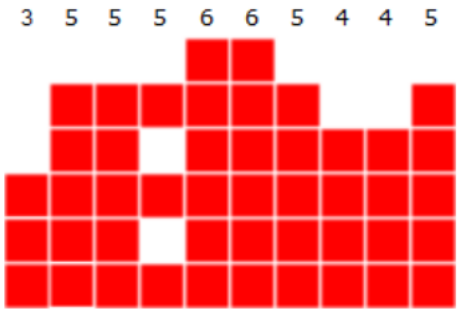
```
    TEST BOARD
   o----------------------o
 0 |           [][]       |
 1 |     [][][][]  []     []|
 2 |     [][]  []  [][][][]|
 3 |[]   [][][][][][][][][]|
 4 |[]   [][]  [][][][][][]|
 5 |[]   [][][][][][][][][]|
   o----------------------o

Number of holes: 4
```

Quiz 7 :

Nous allons implémenter une fonction bumpiness(board) qui est la somme de la valeur absolue de la différence en hauteur des colonnes consécutives. Un exemple a été représenté par l'image 5.1.

Le bumpiness caractérise la variation de la hauteur des colonnes. C'est une valeur que cherche à minimiser l'IA pour avoir une grid la plus plate possible.



$$\text{bumpiness} = |3-5| + |5-5| + \cdots + |4-4| + |4-5| = 6$$

Image 7.1 Schéma indiquant le bumpiness d'un board

Pseudo-code :

```
#Initialiser le compteur 'bump' à 0
#Pour toutes les colonnes sauf la dernière
      #Ajouter à bump l'écart de hauteur entre la colonne et celle d'après
#Retourner 'bump'
```

La fonction retourne :

```
    TEST BOARD
   o--------------------o
 0 |         [][]       |
 1 |   [][][][]   []    []|
 2 |   [][]   []   [][][][]|
 3 |[][][][][][][][][][][]|
 4 |[][][]   [][][][][][]|
 5 |[][][][][][][][][][][]|
   o--------------------o

Bumpiness: 6
```

Quiz 8 :

Après qu'un tetromino soit placé, nous pouvons obtenir un score à ce placement en donnant une liste de poids. Chaque poids est associé à une caractéristique à prendre en compte. Sa valeur est positive et représente la part d'importance d'une caractéristique à l'évaluation du score.

Si la caractéristique présente un désavantage, le score sera soustrait par le poids associé multiplié par la valeur de la caractéristique.
Sinon, le score sera additionné par le poids associé multiplié par la valeur de la caractéristique.
Ainsi, nous pouvons obtenir le score d'un board.

Le board pris pour exemple dans le code est celui de l'image 8.1.

Image 8.1 Schéma du board dont nous allons calculer le score

La fonction retourne :

```
    TEST BOARD
   o--------------------o
 0 |          [][]      |
 1 |   [][][][]   []     []|
 2 |   [][]   []   [][][][]|
 3 |[][][][][][][][][][][]|
 4 |[][][]   [][][][][][]|
 5 |[][][][][][][][][][][]|
   o--------------------o

Score: -56
```

Quiz 9 :

Ce quiz va crée la classe class TetrisAI. Cette classe va lister tous les grids et tetrominos possibles avec next_moves(board, tetromino) puis évaluer le score de chaque possibilité pour retenir le meilleur. De cette manière, l'IA peut sélectionner le tetromino associé au meilleur score, c'est-à-dire le meilleur emplacement du tetromino. Ainsi, l'IA peut déduire quelle action faire pour se rapprocher de cet emplacement.

Chaque action sur le tetromino est associé à un chiffre.

0 : Inactif
1 : Déplacer à gauche
2 : Déplacer à droite
3 : Tourner vers la gauche de 90°
4 : Tourner vers la droite de 90°
5 : Poser tout en bas du board

Pseudo-code :

```
#Chercher la meilleure façon de poser le tetromino
      #Lister toutes les possibilités d'emplacements du tetromino
      #Évaluer le score de chaque possibilité
      #Mettre dans self.target le tetromino posé avec le plus haut score
#Trouver 'action' qui placera le tetromino plus proche du meilleur emplacement
#Réinitialiser le target après avoir posé un tetromino
#Retourner 'action'
```

L'action 3 est retournée par la fonction. Nous pouvons utiliser des print pour mieux comprendre le résultat.

Situation initiale :

```
print(board.add(piece))
print("w=",w)
```

```
 o----------o
0 |          |
1 |   []     |
2 |[][][]    |
3 |          |
4 |    []    |
5 |    []    |
6 |[]  []    |
7 |[]  []  []|
8 |[][][]  []|
9 |[][][]  []|
 o----------o
w= [0.17, 0.55, 0.16, 0.12]
```

Situation après dépôt :

```
#print (grid[i_max])
```

```
 o----------o
0 |          |
1 |          |
2 |          |
3 |          |
4 |  [][]    |
5 |[][][]    |
6 |[][][]    |
7 |[]  []  []|
8 |[][][]  []|
9 |[][][]  []|
 o----------o
```

Nous déduisons que l'IA a besoin de faire les actions 3 puis 5 pour placer le tetromino à l'endroit désiré.

Le code de ce quiz retourne une valeur 3 pour la variable action. Ce qui est la bonne action à faire pour rapprocher le tetromino du meilleur emplacement.

Fonctionnement de l'IA

L'IA possède un moyen de calculer le score d'un board. Dans notre cas, l'IA prend en considération le nombre de lignes complètes, le bumpiness, le nombre de trous et la hauteur totale du board pour le calcul du score. Une liste de poids est aussi pris en compte pour donner plus d'importance à certains paramètres que d'autres.
- Le nombre de lignes complètes du board aura pour effet d'augmenter le score.
- Le bumpiness, le nombre de trous, la hauteur totale du board aura pour effet de diminuer le score.

L'IA aura connaissance du board actuel ainsi que du tetromino qui vient d'apparaître sur le board.
Il est en mesure de lister toutes les possibilités d'emplacement de ce tetromino. Pour ce faire, il associe le type du tetromino aux rotations nécessaires pour obtenir ces différents formes dans l'espace. Puis, pour chaque forme, il va essayer de le poser tout en bas du board de chaque colonne. Ainsi, il obtient une liste exhaustive de tous les placements possibles du tetromino.

Ensuite, il évaluera le score de tous les board de cette liste exhaustive pour ne retenir que le meilleur. Il saura alors quel emplacement de tetromino obtiendra le meilleur score et le placera.

Avec la fonction de jeu donnée ainsi que la liste de poids w = [1,1,1,1], nous pouvons tester l'IA. Mon IA a obtenu le score de 58 soit 58 lignes complètes éliminées avant de perdre.

Description des nouvelles caractéristiques

Quiz 10 :

Ce quiz a pour but d'implémenter une fonction qui retourne la moyenne sur les colonnes, l'écart-type sur les colonnes et le meilleur paramètre de l'échantillon de paramètres élites. Les variables mean et std nous permettront de connaître la moyenne et l'écart-type de chaque poids dans l'échantillon d'élites.

Pseudo-code :

```
#Trouver les indices pour ranger 'scores' dans l'ordre croissant.
#Sélectionner les indices des lignes 'params' avec les n_elites plus grands scores.
#Sélectionner les lignes de 'params' avec les n_elites plus grands scores.
#Calculer la moyenne et l'écart-type des paramètres sur les colonnes.
#Sélectionner le paramètre associé au plus haut score.
```

Si nous utilisons print pour afficher mean, std et best, nous pouvons déduire à l'aide du ok.test que la fonction retourne :

```
[-0.67242271  0.2584151  -0.29266269 -0.90189484 -0.22429414]
[0.41028026 1.05877743 1.19318343 0.35059436 0.85182234]
[-0.56228753 -1.01283112  0.31424733 -0.90802408 -1.4123037 ]
```

Nous pouvons ajouter le code suivant pour mieux visualiser ce que fait cette fonction :

```
print(params)
print(scores)
print(elites)
```

Nous avons l'affichage suivant dans le ok.test associé :

```
params= [[ 0.49671415 -0.1382643   0.64768854  1.52302986 -0.23415337]
 [-0.23413696  1.57921282  0.76743473 -0.46947439  0.54256004]
 [-0.46341769 -0.46572975  0.24196227 -1.91328024 -1.72491783]
 [-0.56228753 -1.01283112  0.31424733 -0.90802408 -1.4123037 ]
 [ 1.46564877 -0.2257763   0.0675282  -1.42474819 -0.54438272]
 [ 0.11092259 -1.15099358  0.37569802 -0.60063869 -0.29169375]
 [-0.60170661  1.85227818 -0.01349722 -1.05771093  0.82254491]
 [-1.22084365  0.2088636  -1.95967012 -1.32818605  0.19686124]
 [ 0.73846658  0.17136828 -0.11564828 -0.3011037  -1.47852199]
 [-0.71984421 -0.46063877  1.05712223  0.34361829 -1.76304016]]

scores= [0.14092422 0.80219698 0.07455064 0.98688694 0.77224477 0.19871568
 0.00552212 0.81546143 0.70685734 0.72900717]

elites= [[-0.23413696  1.57921282  0.76743473 -0.46947439  0.54256004]
 [-1.22084365  0.2088636  -1.95967012 -1.32818605  0.19686124]
 [-0.56228753 -1.01283112  0.31424733 -0.90802408 -1.4123037 ]]
```

En effet, pour chaque ligne de paramètres, nous avons un score associé. Nous pouvons voir que les n_elites (ici, n_elites = 3) meilleurs scores sont `0.80219698` (scores[1] associé à params[1]), `0.98688694` (scores[3] associé à params[3]) et `0.81546143` (scores[7] associé à params[7]).

Les élites sont donc params[1], params[3], params[7]. Pour obtenir la moyenne et l'écart-type de chaque poids, il suffit de faire les calculs sur les colonnes de elites. Pour avoir le meilleur paramètre, il suffit de repérer l'indice du score maximal.

Quiz 11 :

Nous allons implémenter une fonction qui donne la hauteur de l'atterrissage d'un tetromino. L'image 11.1 donne un exemple pour l'appréhender. Le dernier tetromino posé est celui représenté en orange sur l'Image 11.1.

Pour ce quiz, nous allons d'utiliser une fonction donnée qui renvoie la position du centre du tetromino tetromino. Pour obtenir la hauteur d'atterrissage du tetromino, il faut soustraire la taille d'une colonne à la position verticale du centre du tetromino

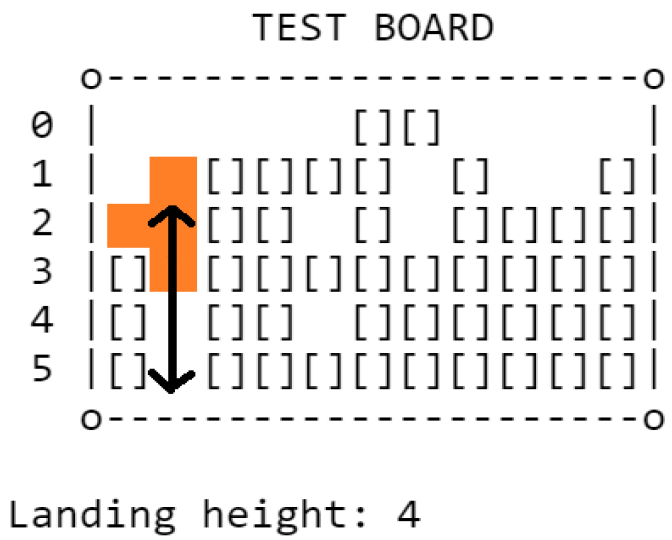


Image 11.1 Schéma du board avec le tetromino dont nous allons calculer la hauteur d'atterrissage

La fonction retourne :

```
    TEST BOARD
   o----------------------o
 0 |           [][]       |
 1 |   [][][][][]  []    []|
 2 |[][][][]  []  [][][][]|
 3 |[][][][][][][][][][]|
 4 |[]  [][]  [][][][][][]|
 5 |[]  [][][][][][][][][]|
   o----------------------o

Landing height: 4
```

Quiz 12 :

Ce quiz a pour but de programmer une fonction qui donne une idée de l'érosion produit par un tetromino qui a complété des lignes. L'érosion sera le produit du nombre de lignes complètes et du nombre de blocs de tetromino impliqués, comme l'indique l'image 12.1.

Nous allons sommer les valeurs du grid sur les lignes pour obtenir le nombre de cases remplies pour chaque ligne. Si une somme est égale à la taille d'une ligne, elle sera considérée complète. Nous allons alors ajouter le nombre de blocs de tetromino impliqués dans le remplissage de cette ligne au nombre total de tetromino impliqué.

Enfin, il suffit de faire le produit du nombre de lignes complètes et du nombre total de blocs de tetromino impliqués pour obtenir l'érosion.

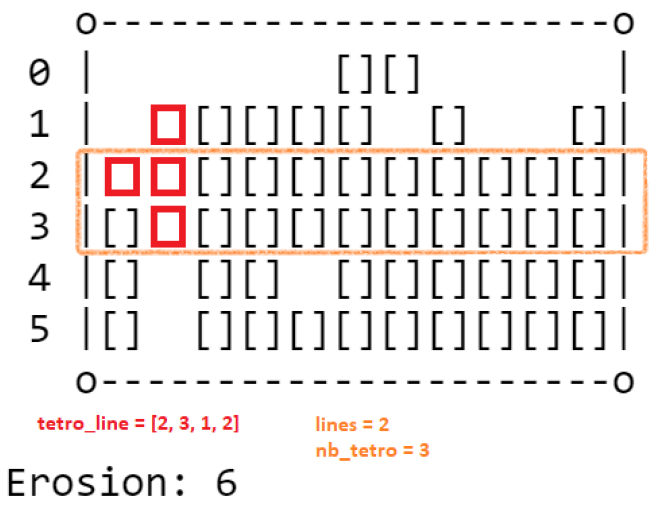


Image 12.1 Schéma indiquant le calcul d'érosion du programme

Pseudo-code :

```
#Initialiser 'lines', 'nb_tetro' et 'erosion' à 0
#Calculer le nombre de cases remplies pour chaque ligne en sommant
#Extraire les lignes auxquelles chaque bloc du tetromino se situent
#Pour toutes les sommes
      #Si cette somme est égale à la taille d'une ligne
            #Incrémenter 'lines'
            #Pour tous les indices de ligne des blocs du tetromino
                  #Si cette indice est égale à la l'indice de la ligne complète
                        #Incrémenter 'nb_tetro'
#Retourner 'lines'*'nb_tetro
```

La fonction retourne :

```
    TEST BOARD
   o----------------------o
 0 |           [][]       |
 1 |   [][][][][]   []    []|
 2 |[][][][][][][][][][][][]|
 3 |[][][][][][][][][][][][]|
 4 |[]   [][]   [][][][][][]|
 5 |[]   [][][][][][][][][]|
   o----------------------o

Erosion: 6
```

Ce qui correspond à l'image 12.1

Quiz 13 :

Ce quiz a pour but de trouver le nombre de transitions sur les lignes (case vide à case remplie ou case remplie à vide) total.
Nous allons utiliser deux boucles for afin de tester si une case est identique à celle située à sa droite. Le cas échéant, nous allons incrémenter le compteur de transitions.
L'extérieur du board sera considéré comme remplie. Donc, si une case sur le board gauche ou droit du board est vide, nous incrémentons aussi le compteur de transitions.

Par exemple, la ligne sélectionnée du board de l'image 13.1 possède 7 transitions.

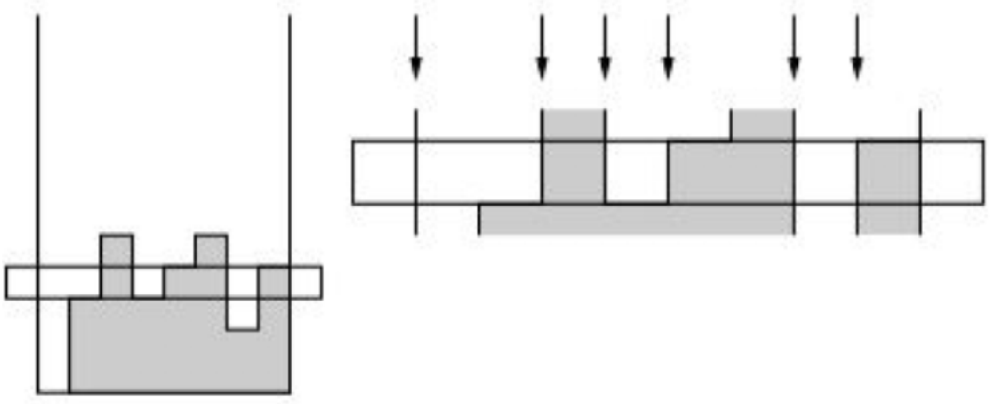


Image 13.1 Schéma indiquant le nombre de transitions d'une ligne d'un board

Pseudo-code :

```
#Initialiser le compteur 'trans' à 0
#Pour toutes les lignes
      #Pour toutes les colonnes sauf la dernière
            #Si la case est différente de celle de droite, incrémenter 'trans'
            #Si la case est vide et est sur le bord gauche/droit, incrémenter 'trans'
#Retourner 'trans'
```

La fonction retourne :

```
    TEST BOARD
   o----------------------o
 0 |           [][]       |
 1 |     [][][][]  []    []|
 2 |     [][][][][][][][][]|
 3 |[]   [][][][][][][][][]|
 4 |[]   [][]  [][][][][][]|
 5 |[]   [][][][][][][][][]|
   o----------------------o

Row transitions: 20
```

Ce qui est correcte, d'après l'image 13.2.

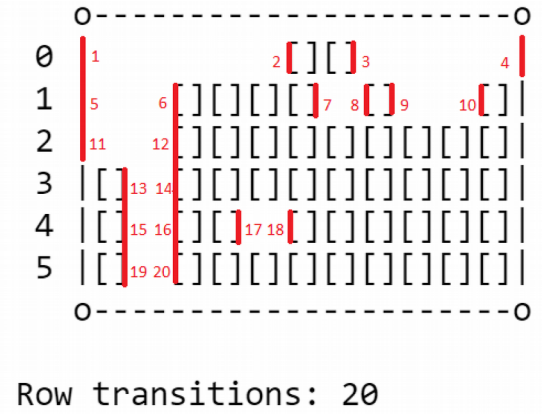


Image 13.2 Schéma indiquant le nombre de transitions du board du programme

Quiz 14 :

Ce quiz est un exercice analogique au quiz 13. Son implémentation est similaire à celle du quiz 13.

Il a pour but de trouver le nombre de transitions sur les colonnes (case vide à case remplie ou case remplie à vide) total.
Nous allons utiliser deux boucles for afin de tester si une case est identique à celle située au dessous Le cas échéant, nous allons incrémenter le compteur de transitions.
L'extérieur du board sera considéré comme remplie. Donc, si une case sur le board haut ou bas du board est vide, nous incrémentons aussi le compteur de transitions.

Pseudo-code :

```
#Initialiser le compteur 'trans' à 0
#Pour toutes les lignes sauf la dernière
      #Pour toutes les colonnes
            #Si la case est différente de celle en dessous, incrémenter 'trans'
            #Si la case est vide et est sur le bord haut/bas, incrémenter 'trans'
#Retourner 'trans'
```

Quiz 15 :

Ce quiz a pour but de trouver la taille cumulée total des pluits du board. La taille cumulée d'un pluits de taille n est égale à la somme de 1 à n. Un pluits est une succession de cases vides telle(s) que les cases à sa gauche et sa à droite sont remplies. Un pluits n'a pas de case remplie au dessus de lui sur la même colonne.

Pour cette exercice, nous allons utiliser la fonction qui liste la hauteur des colonnes pour obtenir la taille du pluits d'une colonne.

- Pour toutes les colonnes sauf la première et la dernière : si la hauteur d'une colonne est inférieur à celle située à sa gauche et à sa droite, la soustraction du minimum de la hauteur de ses colonnes voisins à la sienne donnera la taille du pluits.

- Pour la première colonne : si la hauteur de la seconde colonne lui est supérieur, la soustraction de la hauteur de la deuxième colonne à sa hauteur donnera la taille du pluits.

- Pour la dernière colonne : si la hauteur de l'avant-dernière colonne lui est supérieur, la soustraction de la hauteur de l'avant-dernière colonne à sa hauteur donne sa taille du pluits.

Pour un pluits de taille n, nous allons ajouter la somme d'une liste d'éléments de 1 à n à la taille cumulée total.

Pseudo-code :

```
#Initialiser 'cumulative' à 0
#Pour toutes les colonnes sauf la première et la dernière
      #Si la hauteur d'une colonne est inférieur à celle située à gauche et droite
            #Soustraire le minimum des hauteurs voisines à sa hauteur (noté n)
            #Ajouter la somme d'une liste de 1 à n à 'cumulative'

#Si la hauteur de la seconde colonne est supérieur à la première
      #Soustraire la hauteur de la deuxième colonne à sa hauteur (noté n)
      #Ajouter la somme d'une liste de 1 à n à 'cumulative'
#Si la hauteur de l'avant-dernière colonne est supérieur à la dernière
      #Soustraire la hauteur de l'avant-dernière colonne à sa hauteur (noté n)
      #Ajouter la somme d'une liste de 1 à n à 'cumulative'

#Retourner 'cumulative'
```

La fonction retourne :

```
    TEST BOARD
   o----------------------o
 0 |           [][]       |
 1 |     [][][][]  []    []|
 2 |     [][][][][][][][][]|
 3 |[]   [][][][][][][][][]|
 4 |[]   [][]  [][][][][][]|
 5 |[]   [][][][][][][][][]|
   o----------------------o

Cumulative wells: 6
```

Comme l'indique l'image 15.1.

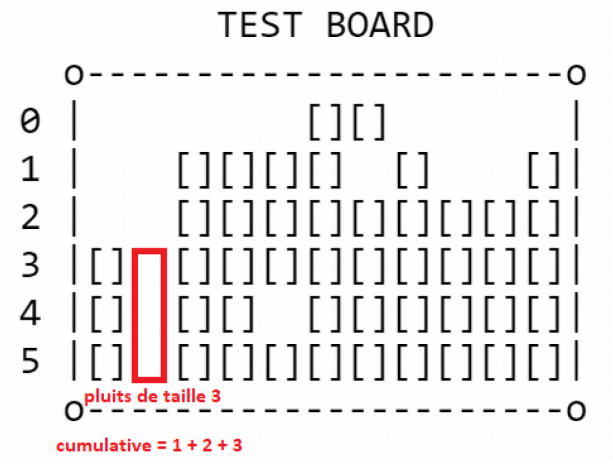


Image 15.1 Schéma indiquant la taille cumulée des pluits du programme

Mais le ok.test ne marche que à 50 %.

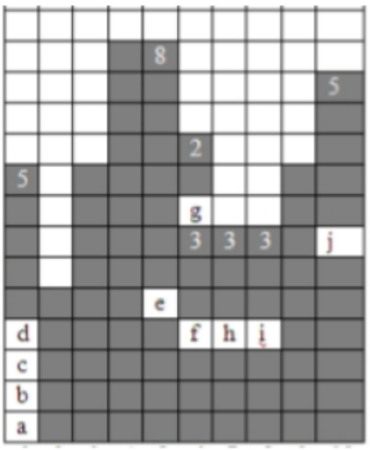
Quiz 16 :

Ce quiz a pour but de trouver la profondeur total des trous du board.

La profondeur d'un trou est le nombre de cases remplies au dessus de celui-ci, sur la même colonne.

Pour calculer la profondeur d'une colonne, nous allons parcourir la colonne en partant du haut. Si la case est remplie, nous allons incrémenter un compteur intermédiaire. Si la case est vide, nous allons ajouter la valeur du compteur intermédiaire au compteur de la profondeur total des trous. Au passage d'une nouvelle colonne, nous réinitialiserons le compteur intermédiaire à 0.

Nous allons additionner la profondeur de chaque trou pour obtenir la profondeur total des trous du board. Dans l'image 16.1, nous avons la profondeur de chaque trou du board. Ainsi, en sommant, nous obtenons une profondeur total des trous de 46.



Hole	Filled Cells
a	5
b	5
c	5
d	5
e	8
f	3+2=5
g	2
h	3
i	3
j	5

Image 16.1 Tableau indiquant la profondeur des trous d'un board

Pseudo-code :

```
#Initialiser les compteurs 'depth' et 'depth_col' à 0
#Pour toutes les colonnes 'i'
      #Pour toutes les lignes 'j'
            #Si la case (i,j) est remplie, incrémenter 'depth_col'.
            #Si la case (i,j) est vide, ajouter 'depth_col' à 'depth'
      #Réinitialiser le compteur 'depth_col' à 0
#Retourner 'depth'
```

La fonction retourne :

```
    TEST BOARD
   o----------------------o
 0 |           [][]       |
 1 |     [][][][]  []     []|
 2 |     [][][][][][][][][]|
 3 |[]   [][][][][][][][][]|
 4 |[]   [][]  [][][][][][]|
 5 |[]   [][][][][][][][][]|
   o----------------------o

Hole depths: 4
```

Quiz 17 :

Ce quiz a pour but de connaître le nombre de lignes avec au moins un trou. Par exemple, l'image 17.1 est un board possédant k lignes avec des trous.

Nous allons utiliser une variable booléen qui vaudra vrai si nous avons découvert un trou sur la ligne, et faux sinon.

Nous allons parcourir chaque ligne, de haut à bas. Pour chaque ligne, si nous n'avons pas encore découvert de trou, nous allons vérifier si la case actuel est un trou. Si oui, nous allons incrémenter le compteur et passer à la ligne suivante. Sinon, nous allons passer à la case suivante s'il ne s'agissait pas du dernier de la ligne. Pour mieux l'appréhender, nous pouvons regarder l'image 17.1.

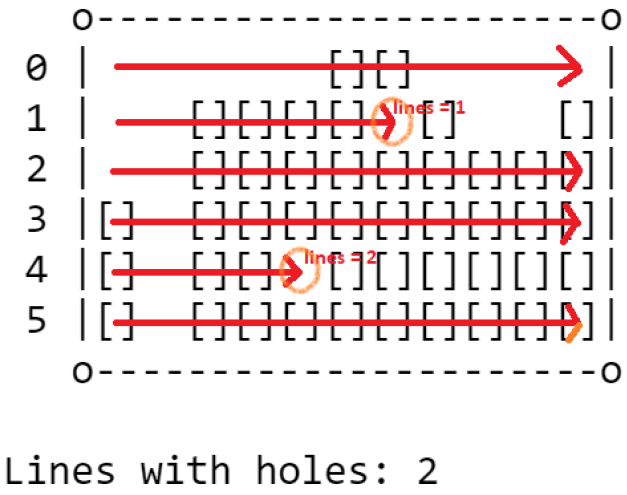


Image 17.1 Schéma indiquant la méthode décrit pour obtenir le nombre de lignes avec trou

Pseudo-code :

```
#Initialiser 'lines', 'j' à 0 et 'have_hole' à False
#Pour toutes les lignes 'i'
      #Tant que have_hole est à False et que 'j' ne déborde pas
            #Si la case (i,j) est un trou
                  #Incrémenter 'lines'
                  #Mettre 'have_hole' à True
            #Sinon
                  #Incrémenter 'j'
      #Réinintialiser 'j' à 0 et 'have_hole' à False
#Retourner 'lines'
```