

DS D'INFORMATIQUE N°3 DU 06/05/2025

Durée : 2h.

- La correction tient compte du soin apporté à la présentation et à la rédaction de la copie. Les bouts de code devront être écrits en langage Python en respectant la syntaxe inhérente à ce langage. On veillera notamment à faire apparaître clairement les indentations qui structurent le code.
 - On pourra utiliser une (ou plusieurs) fonction(s) définie(s) à une question précédente, même si celle-ci n'a pas été résolue.
 - Lorsque l'énoncé impose des contraintes sur le type et la valeur des arguments d'une fonction, on supposera que ces contraintes sont vérifiées lors de l'appel de la fonction. Il n'est pas nécessaire d'ajouter des tests dans la fonction pour vérifier ces contraintes. Exemple : Si l'énoncé demande de définir une fonction d'entête $f(x)$ avec x un nombre réel positif, on supposera que lors de l'appel x est un nombre réel positif. Il n'est pas nécessaire d'ajouter des tests dans la fonction f pour vérifier si x est de type float et, si oui, si il est bien positif.
 - L'usage des calculatrices et de tout document annexe n'est pas autorisé.
-

Tetris Couleurs

On souhaite programmer un jeu de puzzle dont le principe ressemble au jeu de Tetris : des blocs de couleur apparaissent à l'écran en haut d'une aire de jeu et descendent sous l'effet de la « gravité » jusqu'à reposer sur un bloc déjà présent, ou sur le bas de l'aire de jeu. Les nouveaux blocs de couleur apparaissent par groupe de k blocs superposés en une tour verticale appelée « barreau ».

Lors de la descente, le joueur peut déplacer le barreau selon certaines règles. Il peut :

- déplacer le barreau vers la gauche ou vers la droite,
- permuter l'ordre des blocs dans le barreau.

Le but du jeu est de réaliser le plus grand nombre possible d'alignements d'au moins trois blocs de la même couleur. Chaque alignement de trois blocs de la même couleur (horizontalement, verticalement ou en diagonale) donne un point au joueur. Les blocs appartenant à des alignements sont retirés de l'aire de jeu, et les blocs restants sont tassés (ils descendent sous l'effet de la gravité). Les blocs ainsi tassés peuvent à nouveau former des alignements unicolores qui sont à leur tour retirés, et ainsi de suite jusqu'à ce qu'il n'y ait plus aucun alignement unicolore dans l'aire de jeu.

La partie se termine quand l'aire de jeu est trop remplie pour accueillir un nouveau barreau. Le score du joueur est alors la somme des points accumulés lors de la partie.

La partie I contient les principales définitions, les parties II, III sont indépendantes.

Partie I. Initialisation et affichage de l'aire de jeu

L'aire de jeu est représentée par une grille de dimensions `largeur` $\times$ `hauteur`. Dans l'exemple de la Figure 1, la grille est de dimensions 6×12 .

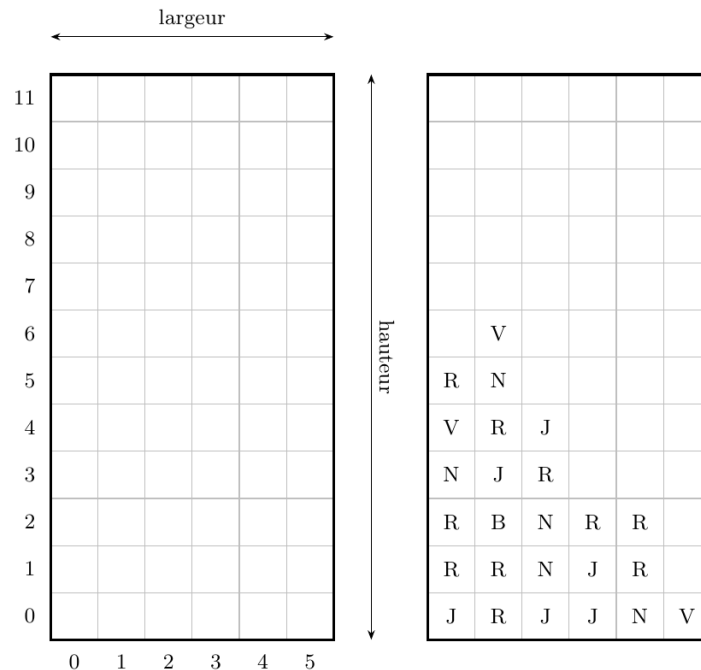


FIGURE 1 : Aire de jeu représentée par une grille (à gauche) et exemple de partie en cours (à droite)

Chaque case de la grille contient une valeur qui représente soit une case vide, soit une couleur. On suppose que les variables Python suivantes sont prédéfinies pour tout le sujet et représentent les couleurs :

- X pour une case vide;
- R, V, B, N, J pour les couleurs.

Le contenu de ces variables (la manière dont elles représentent les couleurs ou le vide) ne nous intéressera pas dans ce sujet. On peut imaginer une chaîne de caractère, ou un emoji carré de couleur, peu importe.

Dans toute la suite on pourra utiliser ces variables à tout instant mais on ne les modifiera jamais.

La Figure 1 à droite montre un exemple d'aire de jeu en cours de partie. Ci-dessous, sa représentation en Python sous la forme d'une liste de listes, ou plus précisément d'une liste de taille `largeur` (ici 6) dont chaque élément est une liste de taille `hauteur` (ici 12) :

```
grille = [ [J, R, R, N, V, R, X, X, X, X, X, X],
            [R, R, B, J, R, N, V, X, X, X, X, X],
            [J, N, N, R, J, X, X, X, X, X, X, X],
            [J, J, R, X, X, X, X, X, X, X, X, X],
            [N, R, R, X, X, X, X, X, X, X, X, X],
            [V, X, X, X, X, X, X, X, X, X, X, X] ]
```

On veillera à respecter l'ordre des dimensions afin que la case de coordonnées (i, j) avec $0 \leq i < \text{largeur}$ et $0 \leq j < \text{hauteur}$ corresponde bien à la valeur `grille[i][j]` dans sa représentation en Python. Ainsi dans `grille[i][j]`, i désigne la colonne et j la ligne, si bien que dans l'exemple ci-dessus la case supérieure droite de la grille est `grille[5][11]`.

Dans toute la suite on parlera de *grille* pour désigner une telle liste de listes contenant uniquement des éléments parmi X, R, V, B, N, J.

1. Écrire une fonction `creerGrille(largeur, hauteur)` qui renvoie une grille de dimensions `largeur × hauteur` dont toutes les cases sont vides.
2. Écrire une fonction `tailleGrille(grille)` qui prend en entrée une grille et renvoie le tuple `(largeur, hauteur)` correspondant aux largeurs et hauteurs selon les définitions ci-dessus.

On souhaite pouvoir afficher à l'écran le contenu de l'aire de jeu. Pour cela, on *présuppose* l'existence des fonctions suivantes :

- `afficheCouleur(c)` qui prend en entrée une variable de couleur (parmi X, R, V, B, N, J) et affiche à l'écran une case de la couleur correspondante, puis déplace le curseur d'affichage d'un cran à droite
- `nouvelleLigne()` qui déplace le curseur d'affichage de l'écran à la ligne suivante (en-dessous).

Ces fonctions ne renvoient rien et se contentent d'effectuer un affichage à l'écran. Tout affichage devra être fait uniquement avec ces procédures, ainsi le `print` de Python est à proscrire dans la suite.

Dans toute la suite, même si le sujet y fera référence, aucune variable globale `largeur` ou `hauteur` a été définie : il faut les définir vous même quand vous en avez besoin. En particulier `largeur` n'est pas fixée à 6, `hauteur` n'est pas fixée à 12.

3. Écrire une fonction `afficheGrille(grille)` qui affiche à l'écran le contenu de l'aire de jeu, encodé dans la liste de listes `grille`. Cette fonction ne doit rien renvoyer, elle se contente d'afficher la grille, de haut en bas, à l'écran, en utilisant les fonctions ci-dessus. On veillera à bien respecter l'orientation verticale de la grille : la ligne apparaissant en bas de l'écran doit correspondre aux éléments `grille[i][0]` pour $0 \leq i < \text{largeur}$.

Partie II. Création et mouvement du barreau.

Au cours d'une partie, le joueur voit apparaître à l'écran un *barreau* consistant en une tour de k blocs colorés, où $k \geq 3$. Le barreau apparaît en pointillé sur les figures. La valeur de k et la couleur des blocs du barreau sont choisies aléatoirement et ne sont pas traitées dans ce problème. La position d'un barreau est donnée à chaque instant par les coordonnées (x, y) de son bloc inférieur. Dans la suite, on supposera toujours que les coordonnées (x, y) définissent une case dans la grille (c'est-à-dire $0 \leq x < \text{largeur}$ et $0 \leq y < \text{hauteur}$) et que $y + k \leq \text{hauteur}$ (et donc $k \leq \text{hauteur}$).

Un barreau de taille k peut naître au sommet d'une colonne si les k cases supérieures de cette colonne sont vides (voir Figure 2). Si aucune colonne n'a ses k cases supérieures vides, la partie s'arrête.

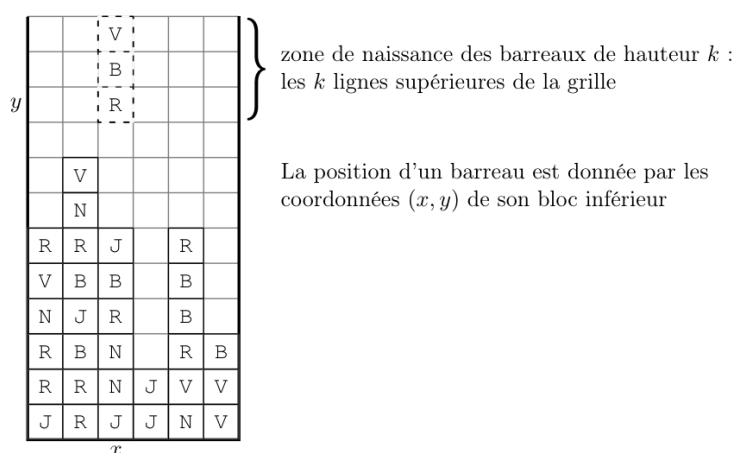


FIGURE 2 : Zone de naissance des barreaux

Dans la suite, aucune variable globale k n'est définie au préalable. En particulier, k n'est pas fixé à 3.

- Étant donné une case de coordonnées (x, y) et un entier naturel k , compléter l'instruction suivante permettant de mettre en mémoire dans une liste `bloc` les k couleurs stockées en (x, y) , $(x, y + 1)$, $\dots$, $(x, y + k - 1)$, dans cet ordre. On supposera que tous les indices sont valides.

`bloc = grille[ ][ : ]`

- Écrire une fonction `grilleLibre(grille, k)` qui renvoie **True** si dans au moins une colonne de la grille, les k premières cases (en partant du haut de la grille) sont vides, et renvoie **False** sinon. On ne fait pas d'hypothèse sur le contenu de la grille.
- Donner un ordre de grandeur du nombre d'opérations élémentaires effectuées par votre fonction. On pourra répondre en fonction de k , largeur et hauteur.

En l'absence d'intervention du joueur, le barreau descend d'une case à chaque unité de temps. La descente s'arrête dès que le barreau atteint le bas de la grille, ou rencontre un bloc déjà présent (le passage du temps est illustré à la Figure 3).

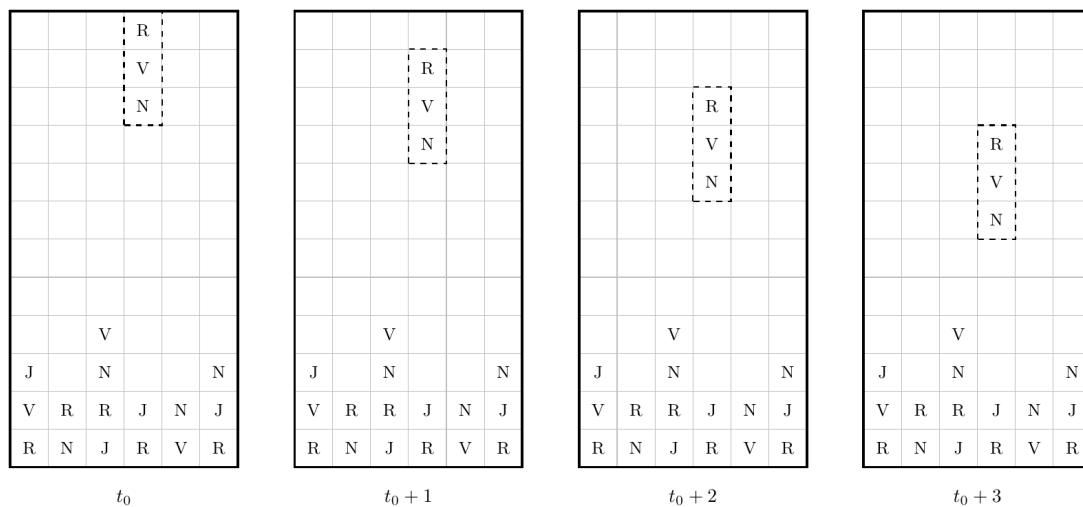


FIGURE 3 : Descente du barreau

- Écrire une fonction `descente(grille, x, y, k)` qui prend en arguments une grille et les coordonnées (x, y) du bloc inférieur d'un barreau de hauteur k (déjà entièrement dans la grille), et modifie la grille en faisant descendre le barreau d'une case. Si le barreau ne peut pas descendre, la grille n'est pas modifiée. Cette fonction ne doit rien renvoyer mais se contenter de travailler sur la variable `grille` par effet de bord.

Le joueur peut déplacer le barreau d'une colonne vers la gauche ou vers la droite en utilisant les flèches du clavier. Le déplacement du barreau vers la droite n'est possible que si le barreau n'est pas contre le bord droit de la grille et que les k cases se trouvant à sa droite sont toutes vides. Symétriquement, le déplacement du barreau vers la gauche n'est possible que si le barreau n'est pas contre le bord gauche de la grille et que les k cases se trouvant à sa gauche sont toutes vides (Figure 4).

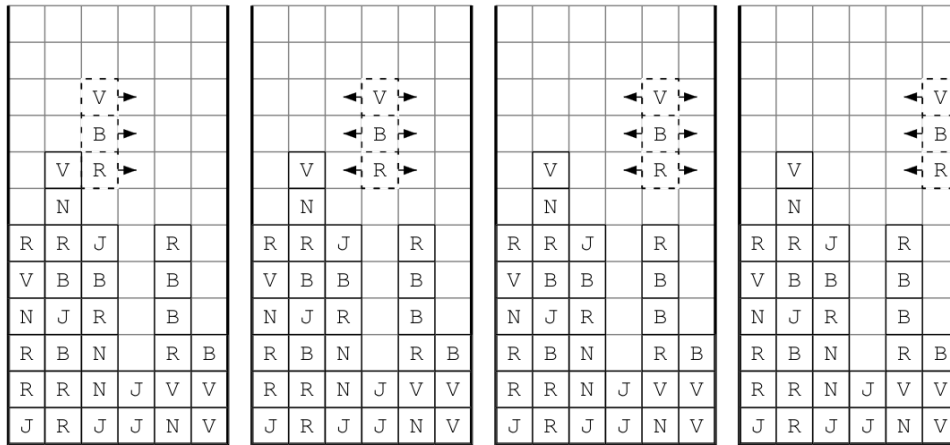


FIGURE 4 : Déplacements possibles du barreau

8. Écrire une fonction `deplacerBarreau(grille, x, y, k, direction)` qui prend en argument une grille et les coordonnées (x, y) du bloc inférieur d'un barreau de hauteur k (déjà entièrement dans la grille), et qui modifie la grille en déplaçant le barreau d'une case vers la gauche si $direction = -1$ ou vers la droite si $direction = 1$. Si le déplacement du barreau n'est pas possible, la grille reste inchangée. Cette fonction ne doit rien renvoyer mais se contenter de travailler sur la variable `grille` par effet de bord.

Indice : utiliser des tranches comme en question 4.

Le joueur peut aussi modifier l'ordre des blocs dans le barreau par une permutation circulaire qui fait remonter chaque bloc d'une case, sauf le bloc le plus haut qui redescend à la place du bloc le plus bas (Figure 5).

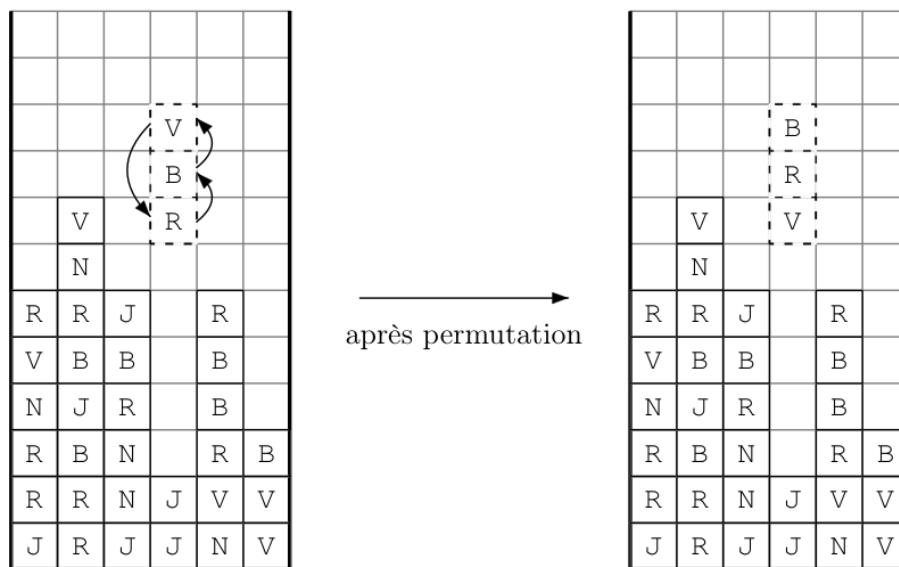


FIGURE 5 : Permutation circulaire du barreau

9. Écrire une fonction `permuterBarreau(grille, x, y, k)` qui modifie la grille en effectuant, comme décrit ci-dessus, une permutation circulaire des couleurs du barreau de hauteur k (non nécessairement égal à 3) dont le bloc inférieur est en position (x, y) dans la grille donnée en argument. Cette fonction ne doit rien renvoyer mais se contenter de travailler sur la variable `grille` par effet de bord.

Partie III. Détection des alignements et calcul du score

Lorsqu'un barreau ne peut plus descendre, le joueur gagne des points si des alignements d'au moins trois blocs de la même couleur sont réalisés dans la grille. Les alignements peuvent être réalisés sur une ligne, sur une colonne, ou en diagonale. Notre but est maintenant de détecter les alignements unicolores et de calculer le score du joueur.

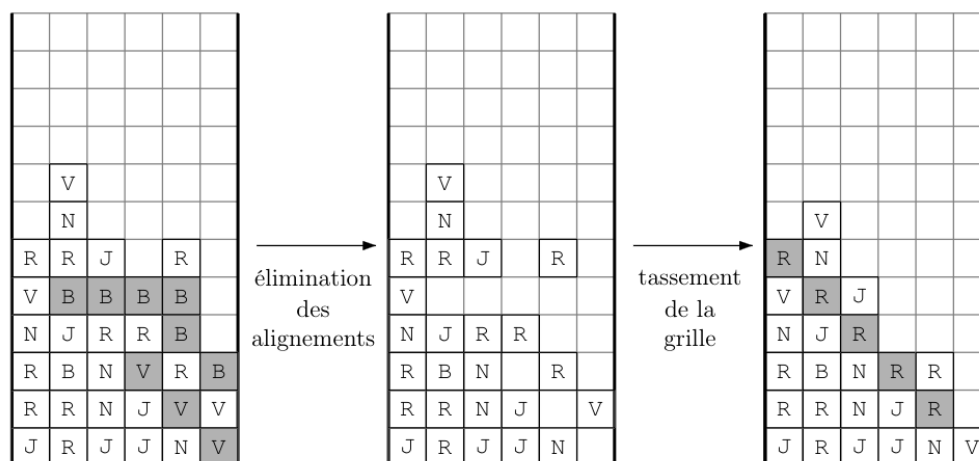


FIGURE 6 : Élimination des alignements unicolores et tassement de la grille.

Chaque alignement unicolore de longueur $m \geq 3$ donne $m - 2$ points au joueur. Cette règle ne s'applique que si l'alignement de longueur m n'est pas lui-même inclus dans un alignement de longueur plus grande que m , donc on ne prend en considération que les alignements de longueur maximale pour calculer le score. Par exemple l'alignement horizontal de quatre blocs de couleur B dans la Figure 6 donne 2 points, ceux en diagonale de trois blocs de couleur B et V donnent chacun 1 point. Le bloc B de coordonnées (3, 4) compte à la fois pour l'alignement horizontal et l'alignement en diagonale. Le joueur marque donc 4 points dans cette configuration.

Tous les blocs appartenant à de tels alignements sont ensuite *simultanément* retirés de la grille et remplacés par des cases vides (deuxième grille de la Figure 6). Les blocs restants sont ensuite tassés, c'est-à-dire qu'ils descendent du maximum possible de cases. Il se peut que de nouveaux alignements se forment après le tassement de la grille (comme les cinq blocs de couleur R dans la troisième grille de la Figure 6). Ces nouveaux alignements donnent à nouveau des points au joueur (selon le même barème que précédemment) et le même processus d'élimination des alignements et de tassement de la grille est réalisé. Ce processus se poursuit jusqu'à ce qu'il n'y ait plus d'alignement unicolore de longueur $m \geq 3$ dans la grille. Dans l'exemple, le joueur marque au total 7 points.

10. Calculer le nombre de points marqués par le joueur s'il laisse descendre complètement le barreau dans l'aire de jeu ci-dessous. Justifier votre réponse.

				B		
				R		
				J	N	
				R	N	
				R		
	B	B		N	J	
	N	R		V	N	
N	R	R		R	V	
N	B	J	B	V	V	
V	N	J	B	V	J	

Pour réaliser la fonction qui va détecter et comptabiliser les alignements unicolores de la grille, on va d'abord construire une fonction qui réalise cela sur une liste simple (à une dimension donc).

11. Écrire une fonction `detecteAlignement(rangee)` qui prend en argument une liste `rangee` non vide contenant des valeurs parmi X, R, V, B, N, J et qui renvoie un tuple (`marking`, `score`) de deux éléments :
- `marking` est une liste de la même taille que `rangee` contenant des booléens, et telle que `marking[i] = True` si et seulement si `rangee[i]` appartient à un alignement unicolore de longueur au moins 3. (Cette liste nous servira plus tard quand il faudra supprimer des couleurs lors du calcul des points).
 - `score` est le nombre de points obtenus par le joueur pour les alignements présents dans `rangee` (selon le barème donné plus haut).

Par exemple, pour la liste

```
rangee = [B, R, R, R, R, J, J, J, X, X, X]
```

la fonction `detecteAlignement(rangee)` renvoie :

- `marking = [False, True, True, True, True, True, True, True, False, False, False]`
- `score = 3`.

On demande que lors du traitement, la fonction `detecteAlignement(rangee)` n'accède qu'une seule fois à chaque élément de la liste `rangee`.

Indice : parcourir la liste et détecter les changements de couleur ou l'arrivée en bout de liste.

Pour détecter les alignements unicolores de la grille et comptabiliser les points marqués, nous allons explorer toutes les lignes, colonnes et diagonales de la grille. Pendant le traitement, on va conserver une copie de travail de la grille dans laquelle on remplacera les cases appartenant à un alignement unicolore par des cases vides.

12. Si l'on veut copier une grille `grille` dans une variable `g`, expliquer en quoi l'instruction

```
g = grille
```

est problématique et proposer une correction.

13. Écrire une fonction `scoreRangee(grille, g, i, j, dx, dy)` qui prend en arguments

- une grille `grille`
- une grille `g` de mêmes dimensions que `grille`
- les coordonnées (i, j) d'une case dans la grille
- une direction (dx, dy) donnée par $dx, dy \in \{-1, 0, 1\}$.

et qui

- construit une liste `rangee` à une dimension contenant les valeurs des cases de `grille` dont les coordonnées sont (i, j) , $(i+dx, j+dy)$, $(i+2*dx, j+2*dy)$, ...
- puis utilise la fonction `detecteAlignement(rangee)` de la Question 11 pour détecter les alignements unicolores dans la liste `rangee`, déterminer le nombre de points marqués et mettre à jour la grille `g` pour que les cases appartenant à un alignement unicolore dans `grille` soient vides dans `g`,
- renvoie le nombre de points marqués.

Attention : la liste `grille` ne doit par contre pas être modifiée.

14. Écrire une fonction `effaceAlignement(grille)` qui prend en argument une grille et renvoie un tuple (`g`, `score`) de deux éléments :

- `g` est la grille mise à jour où tous les blocs appartenant à un alignement unicolore sont remplacés par des cases vides.
- `score` est le nombre total de points obtenus par le joueur pour les alignements présents dans la grille.

La grille `grille` ne doit pas être modifiée.

Indice : commencer par soigneusement énumérer sur votre copie toutes les rangées horizontales, verticales, et diagonales montantes ou descendantes à considérer, et ce sans doublon.

15. Écrire une fonction `tassementGrille(grille)` qui modifie la grille donnée en argument en effectuant le tassement de ses cases non vides. Cette fonction ne doit rien renvoyer.
16. Écrire une fonction `calculScore(grille)` qui :
 - effectue une copie de la grille `grille` dans une variable `g`,
 - met à jour la grille `g` après élimination des alignements et tassement, répétés jusqu'à ce que `g` ne contienne plus aucun alignement unicolore de longueur $m \geq 3$,
 - et qui renvoie le tuple `(g, scoreTotal)` où `scoreTotal` est le nombre total de points marqués par le joueur pour les alignements éliminés de la grille.