Lycée Michelet, classes de PC* et de PSI Lundi 14 octobre 2019

Informatique

Devoir surveillé n°1 bis

Durée 2h — sans ordinateur ni calculatrice

Dans tout l'énoncé, le résultat renvoyé par un code donné est noté : `code` → `résultat`
Toute fonction évoquée dans l'énoncé pourra être utilisée pour répondre aux questions suivantes. L'efficacité, la clarté (penser à commenter les fonctions) et la simplicité du code seront pris en compte dans la notation.

Questions indépendantes

Q A Écrire une fonction `multiple(n,L)` d'argument un entier `n` et un liste d'entiers `L`, qui renvoie `True` si `n` est multiple de chacun des éléments de `L`, et qui renvoie `False` sinon.

Q B Écrire une fonction `f1(L,x)`, où l'argument `L` est une liste, qui renvoie la dernière position où `x` apparaît dans `L`, et qui renvoie `False` si `x` n'est pas dans `L`. Par exemple :
`f1([1,2,3,3,4,5,3,6],1)`→ `0`
`f1([1,2,3,3,4,5,3,6],3)`→ `6`
`f1([1,2,3,3,4,5,3,6],8)`→ `False`

Q C (a) Écrire une fonction `calcule(u0,f,n)`, où `u0` est un flottant, où `f` une fonction définie sur $\mathbb{R}_+$ et à valeurs dans $\mathbb{R}$, et où `n` un entier naturel. Cette fonction renvoie, si elle est bien définie, la somme $\sum_{k=0}^{\texttt{n}} u_k$ des $n+1$ premiers termes de la suite définie par récurrence par $u_0 =$`u0` et $u_{k+1} = f(u_k)$ pour tout k. Cette fonction renvoie `False` si cette somme n'est pas définie.

(b) En utilisant cette fonction, donner un script qui permet de calculer $\sum_{k=0}^{1000} u_k$, où (u_n) est la suite définie par récurrence par $u_0 = 2019$ et $u_{n+1} = {u_n}^2 + 1$.

Q D Dans cette question, `T` désigne une matrice carrée réelle d'ordre n représentée en `Python` par une liste formée de n listes de chacune n flottants (appelés « coefficients » de la matrice `T`).

(a) Écrire une fonction `ecart(T)` qui renvoie la somme $\sum_{0\leq i,j\leq n-1} (\texttt{T[i][j]-T[0][0]})^2$.
Par exemple `ecart([[1,2],[3,4]])`→ `14`.

(b) Écrire une fonction `suivant(T)` qui ne renvoie rien (c'est-à-dire que c'est une procédure) mais qui modifie `T` en remplaçant chaque coefficient de `T` par la moyenne de tous les autres coefficients de `T` (hormis lui-même). Par exemple `[[3,6],[9,12]]` est transformée en `[[9,8],[7,6]]`.

(c) Écrire une fonction `limite(T, seuil)` qui modifie `T` avec la fonction `suivant` autant que fois que nécessaire pour que la valeur de `ecart(T)` soit inférieure à `seuil`. Cette fonction renvoie sous forme de liste la dernière valeur de `T[0][0]` ainsi calculée et le nombre de fois où l'on a effectué la fonction `suivant`.

Problème : Tétris couleur

On souhaite programmer un jeu dont le principe ressemble à celui du jeu de Tétris : des « barreaux » formés de blocs colorés apparaissent à l'écran en haut d'une aire de jeu et descendent sous l'effet de la « gravité » jusqu'à reposer sur les blocs déjà présents ou sur le bas de l'aire de jeu. Un « barreau » est un groupe de k blocs carrés colorés placés verticalement les uns sur les autres. Lors de la descente, le joueur peut effectuer les opérations suivantes sur le barreau :

- déplacer le barreau vers la gauche ou vers la droite ;
- modifier l'ordre des blocs dans le barreau ;
- faire descendre le barreau « rapidement ».

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 formé donne des points au joueur. Puis les blocs appartenant à des alignements sont retirés de l'aire de jeu et les blocs restants se tassent vers le bas sur les places libérées, sous l'effet de la « gravité » ; les blocs tassés peuvent à nouveau former des alignements unicolores qui sont à leur tour retiré du jeu, et ainsi de suite jusqu'à ce qu'il n'y ait plus aucun alignement unicolore dans l'aire de jeu.

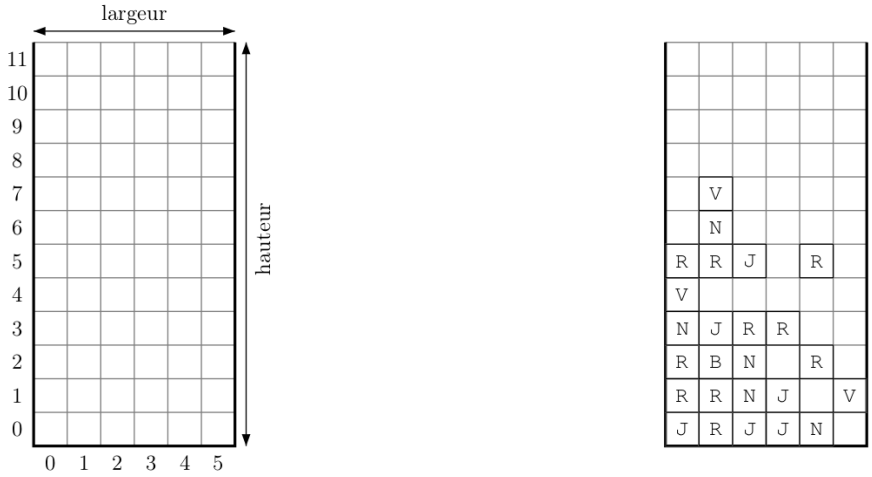
La partie se termine quand l'aire de jeu est trop remplie pour accueillir un nouveau barreau. Le score du joueur est alors la somme des points accumulés lors de la partie.

On rappelle que l'on peut récupérer directement les valeurs contenues dans une liste par une instruction du type `[a,b,c]=L` ; si `L` était la liste `[1,2,3]`, alors les variables `a,b` et `c` valent respectivement `1,2` et `3`. Pour la clarté des programmes proposés, il est demandé d'utiliser cette syntaxe.

1 Initialisation et affichage de l'aire de jeu

L'aire jeu est représentée par une grille de dimension `largeur`×`hauteur`. Dans l'exemple de la figure 1(a), la grille est de dimension 6×12.

FIGURE 1 — L'aire de jeu



(a) Aire de jeu représentée par une grille.

(b) Grille en cours de partie.

Chaque case de la grille contient une valeur qui représente soit une case vide, codée par la chaîne de caractère `"VIDE"`, soit une couleur, codée par une des chaînes de caractère `"R","V","B","N"` ou `"J"`. Une grille sera représentée en `Python` par une liste de listes. La figure 1(b), qui donne un exemple d'aire de jeu en cours de partie, de taille 6×12 sera représentée par la liste de 6 listes de longueur 12 suivante :

```
grille = [["J",    "R",    "R",    "N",    "V",    "R",    "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE"],
          ["R",    "R",    "B",    "J",    "VIDE", "R",    "N",    "V",    "VIDE", "VIDE", "VIDE", "VIDE"],
          ["J",    "N",    "N",    "R",    "VIDE", "J",    "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE"],
          ["J",    "J",    "VIDE", "R",    "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE"],
          ["N",    "VIDE", "R",    "VIDE", "VIDE", "R",    "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE"],
          ["VIDE", "V",    "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE", "VIDE"]]
```

Ainsi la valeur de la case supérieure droite de la grille est `grille[5][11]`. On veillera à bien respecter l'ordre des dimensions afin que la case de coordonnées (i, j), avec $0 \leq i <$ `largeur` et $0 \leq j <$ `hauteur` (cf. figure 1(a)) corresponde bien à la valeur `grille[i][j]` dans sa représentation en `Python`.

Q1 Écrire une fonction `creerGrille(largeur, hauteur)` qui renvoie une grille de dimensions `largeur`×`hauteur` dont toutes les cases sont vides.

On souhaite pouvoir afficher à l'écran le contenu de l'aire de jeu à l'aide de l'instruction `print`. Par exemple la grille de la figure 1(b) sera affichée comme ci-dessous :

```
 V
 N
RRJ R
V
NJRR
RBN R
RRNJ V
JRJJN
```

On rappelle qu'après chaque utilisation de `print`, le curseur passe à la ligne suivante (pour le prochain appel de `print`).

Q2 (a) Écrire une fonction `ligne2chaine(grille,i)` qui renvoie la ligne `i` de la grille `grille` sous la forme d'une chaine de caratères où les cases vides sont codées par la chaîne `" "` (un espace). Par exemple, avec la grille présentée ci-avant, `ligne2chaine(grille,2)`→ `"RBN R "`

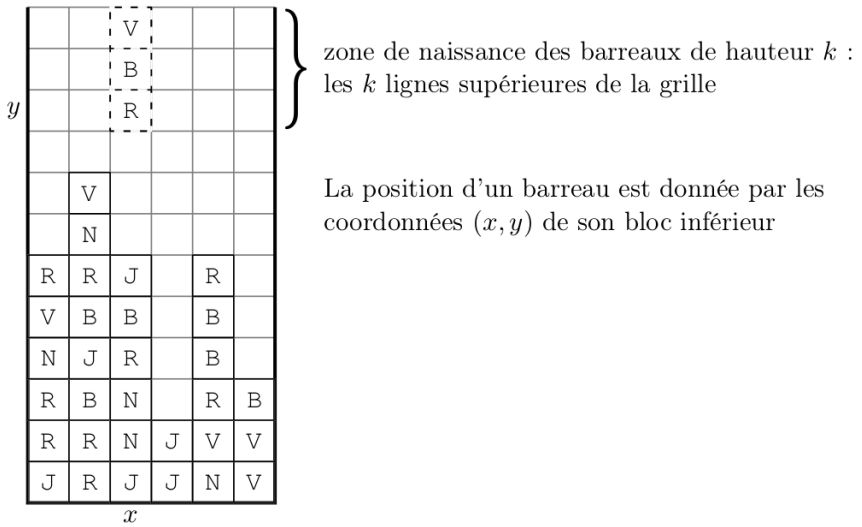
(b) Écrire une procédure `afficheGrille(grille)` qui affiche à l'écran le contenu de l'aire de jeu encodée dans la variable `grille`. On veillera à bien respecter l'orientation *verticale* de la grille, c'est-à-dire que la ligne du bas de l'écran devra correspondre aux éléments `grille[i][0]` pour $0 \leq i <$ `largeur`.

2 Création et mouvement du barreau

Au cours de la partie, le joueur voit apparaître à l'écran un *barreau* consistant en une tour de k blocs où $k \geq 3$; ce barreau apparaît en pointillés sur les figures. La valeur de k et les couleurs des blocs du barreau sont choisies aléatoirement. La position du barreau est donnée 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 que $0 \leq x <$ `largeur` et $0 \leq y <$ `hauteur`) et que $y + k <$ `hauteur`.

Un barreau peut naître au sommet d'une colonne ayant ses k cases supérieures vides (voir figure 2). Si aucun colonne n'a ses k cases supérieures vides, la partie s'arrête.

FIGURE 4 — Zone de naissance des barreaux

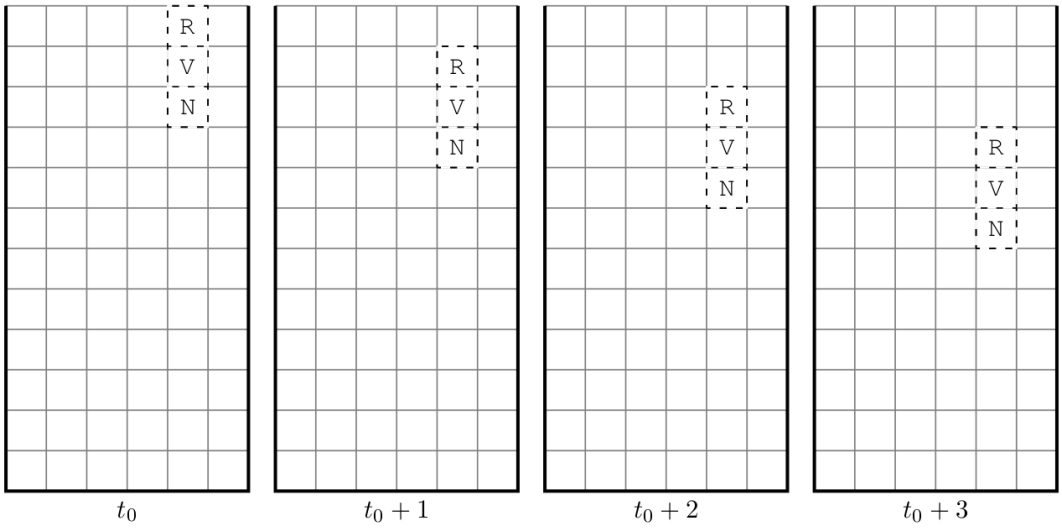


Q3 Écrire une procédure `grilleLibre(grille,k)` qui renvoie `True` si dans au moins une colonne de la grille `grille`, les k dernières cases du haut sont vides, et qui renvoie `False` sinon. On ne fait pas d'hypothèse sur le contenu des grilles (en particulier la grille n'est pas nécessairement tassée).

Quelle est la complexité de votre fonction ?

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 base de la grille, ou rencontre un bloc déjà présent (évolution dans le temps illustré à la figure 3).

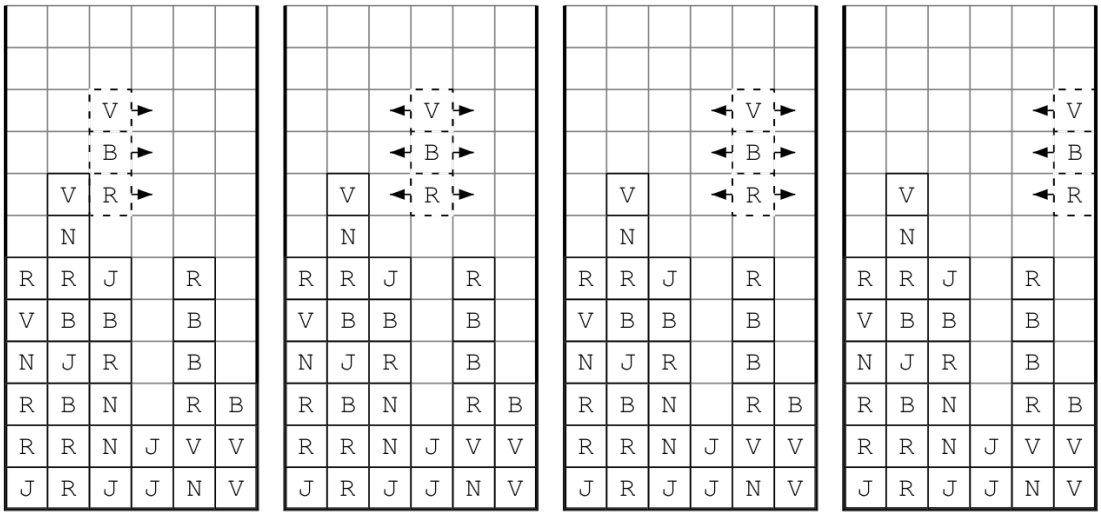
FIGURE 3 — Descente du barreau



Q4 Écrire une procédure `descente(grille,x,y,k)` qui prend en argument une grille `grille` et les coordonnées `x,y` d'un barreau de hauteur `k`, et modifie la grille en faisant descendre le barreau d'une case. Si le barreau ne peut pas descendre, la grille `grille` n'est pas modifiée par cette fonction.

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 droite sont toutes vides. Et de même à gauche (voir figure 4).

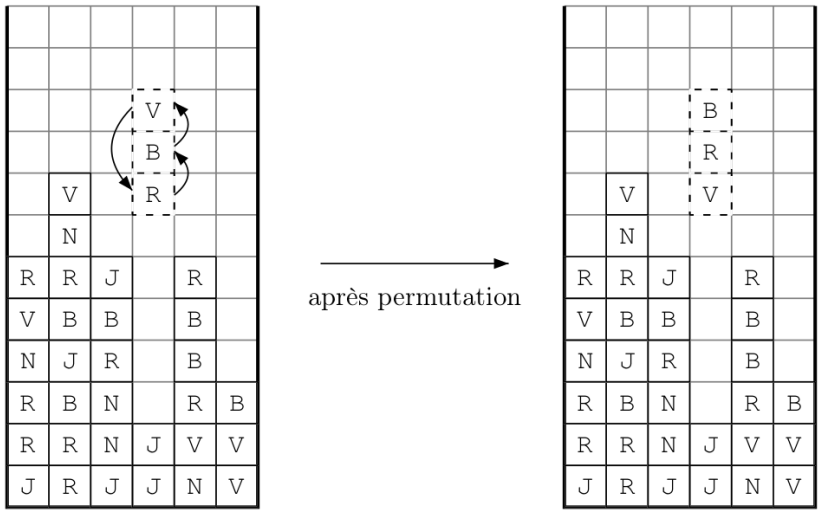
FIGURE 4 — Déplacements possibles du barreau



Q5 Écrire une procédure `deplacerBarreau(grille,x,y,k,direction)` qui prend en argument une grille `grille`, les coordonnées `x,y` d'un barreau de hauteur `k` et un entier `direction` qui vaut `1` ou `-1`, et modifie la grille en déplaçant le barreau d'une case vers la droite si `direction` vaut `1` ou vers la gauche si `direction` vaut `-1`. Si le déplacement n'est pas possible, la grille `grille` n'est pas modifiée.

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

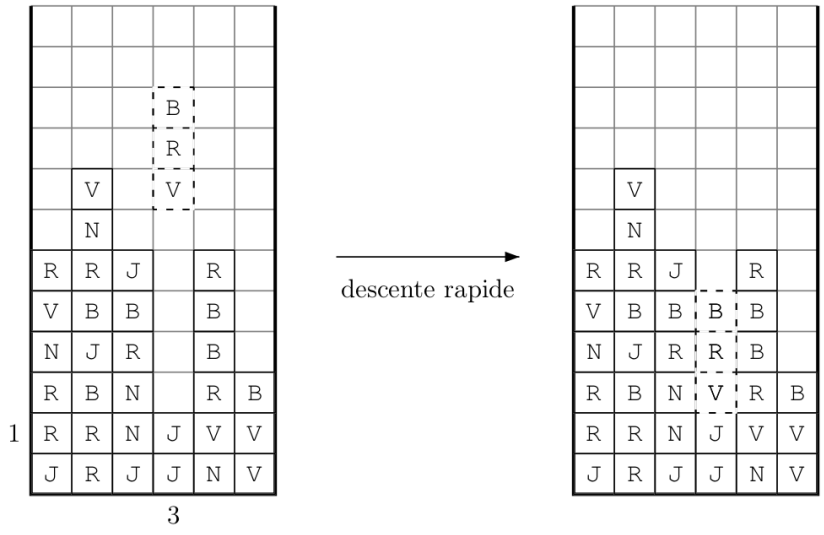
FIGURE 5 — Permutation circulaire du barreau



Q6 Écrire une procédure `permuterBarreau(grille,x,y,k)` qui prend en argument une grille `grille`, les coordonnées `x,y` d'un barreau de hauteur `k` et qui modifie la grille en effectuant la permutation décrite ci-dessus.

Enfin le joueur peur faire descendre le barreau « rapidement », ce qui signifie que le barreau descend du nombre maximum possible de cases pour venir se poser au dessus de la première cases non vide (voir figure 6) ou sur le fond de la grille si toutes les cases situées sous le barreau sont vides. Dans l'exemple de la figure 6, la première case non vide située sous le barreau a pour coordonnées $(3, 1)$.

FIGURE 6 — Descente rapide du barreau



Q7 Écrire une procédure `descenteRapideBarreau(grille,x,y,k)` qui prend en argument une grille `grille`, les coordonnées `x,y` d'un barreau de hauteur `k` et qui modifie la grille en effectuant en faisant descendre le barreau « rapidement ». On demande que la fonction ait une complexité en O(k+`hauteur`) — et non en O(k ×`hauteur`).

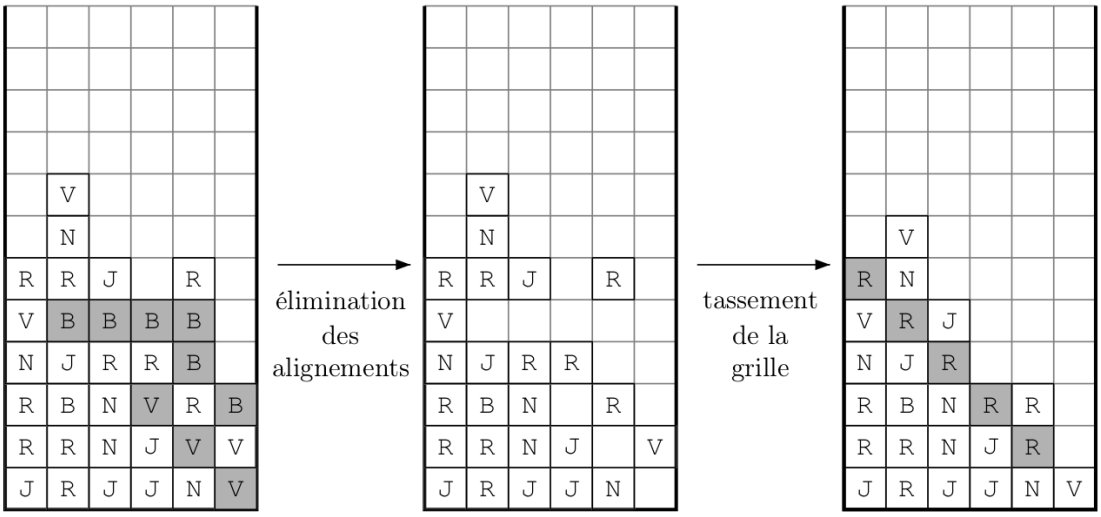
3 Détection des alignements et calcul du score

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, une colonne ou en diagonale. Notre but est maintenant de détecter les alignements et de calculer le score du joueur.

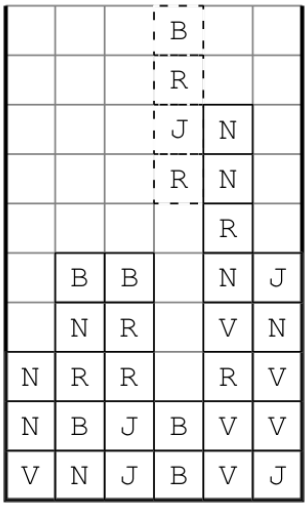
Chaque alignement unicolore de longueur $m \geq 3$ donne $m - 2$ points au joueur. Cette règle ne s'applique que si l'alignement de longueur m n'est pas lui-même inclus dans un alignement de longueur plus grande que m, c'est-à-dire que l'on ne prend en compte que les alignements de longueur maximale pour calculer le score. Par exemple, sur la figure 7, l'alignement horizontal de quatre blocs donne deux points, ceux en diagonale de trois blocs de couleurs respectives `B` et `V` donnent chacun un point. Le bloc de couleur `B` de coordonnées $(3, 4)$ compte à la fois pour deux alignements. 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 7). Les blocs restant sont ensuite tassés, c'est-à-dire qu'il descendent au maximum. Il se peut que de nouveaux alignements se forment après tassement de la grille (comme les cinq blocs de couleur `R` de la troisième grille de la figure 7). Ces nouveau alignements donnent à nouveau des points au joueur selon la même règle que précédemment puis le même processus d'élimination et de tassement de la grille s'effectue à nouveau. Ce processus se poursuit jusqu'à qu'il n'y ait plus d'alignement unicolore de longueur $m \geq 3$ dans la grille. Dans l'exemple de la figure 7, le joueur marque au total 7 points.

FIGURE 7 — Élimination des alignements unicolore et tassement de la grille



Q8 Calculer le nombre de points marqués par le joueur s'il laisse descendre le barreau dans l'aire de jeu ci-dessous. Justifier votre réponse.



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 un tableau simple (à une dimension).

Q9 Écrire une fonction `detecteAlignement(rangee)` qui prend en argument un tableau `rangee` non vide à une dimension contenant des valeurs prises dans l'ensemble {`"VIDE"`,`"R"`,`"V"`,`"B"`,`"N"`,`"J"`}, et qui renvoie une liste `[marking, score]` de deux éléments :

- `marking` est un tableau de la même taille que `rangee` contenant des booléens, tel que `marking[i]` vaut `True` si et seulement si `rangee[i]` appartient à un alignement unicolore de longueur au moins 3.
- `score` est le nombre de points obtenus par le joueur pour les alignements présents dans `rangee` (selon la règle donnée plus haut).

Par exemple :

```
detecteAlignement(["B","R","R","R","R","J","J","J","VIDE","VIDE","VIDE"])
```

$\rightarrow$ `[[False,True,True,True,True,True,True,True,False,False,False],3]`

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

Les alignements obtenus après tassement du tableau `rangee` ne seront pas pris en compte.

Indication : parcourir le tableau et détecter les changements de couleur.

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. Dans les questions suivantes, on suppose que `g` est cette copie de travail.

Q10 (a) Dans le cas particulier où `largeur` vaut 6 et `hauteur` vaut 12, donner le nombre et la liste de toutes les rangées (lignes, les colonnes ou diagonales) à parcourir, sans doublon. Chaque rangée sera donnée sous la forme `i,j,dx,dy`, où `i,j` sont les cooordonnées d'une case du bord de l'aire de jeu, et où `dx,dy` $\in \{-1, 0, 1\}$ sont choisis de telle sorte que la rangée soit formée des cases dont les coordonnées sont `(i,j)`, `(i+dx,j+dy)`, `(i+2*dx,j+2*dy)`, etc.

(b) Écrire une fonction `scoreRangee(grille,g,i,j,dx,dy)` qui prend en argument une grille `grille`, une grille `g` de mêmes dimensions, et une rangée `i,j,dx,dy` (comme expliqué juste avant). Cette fonction utilise la fonction `detecteAlignementRangee(rangee)` de la question 9 pour détecter les alignements unicolore dans le tableau `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`. La grille `grille` ne doit pas être modifiée. La fonction renvoie le nombre de points marqués.

Q11 (a) Écrire une fonction `effaceAlignement(grille)` qui prend en argument une grille `grille` et renvoie une liste `[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.

(b) Donner la complexité de votre fonction.

Q12 Écrire une procédure `tassement(grille)` qui modifie la grille `grille` donnée en argument en effectuant le tassement de ses cases non vides.

Q13 Écrire une fonction `calculeScore(grille)` qui met à jour la grille `grille` après élimination des alignements et tassement répétés jusqu'à ce que la grille ne contienne plus aucun alignement unicolore de longueur $m \geq 3$, et qui renvoie le nombre total de points marqués par le joueur pour les alignements éliminés de la grille.

Fin du devoir

Lycée Michelet, classes de PC* et de PSI

Informatique

Corrigé du devoir surveillé n°1bis

Q A

```
def multiple(n,L):
    for j in L:
        if n%j!=0:
            return False
    return True
```

Q B

```
def f1(L,x):
    for i in range(len(L)-1,-1,-1): #parcours de la liste L l'envers
        if L[i]==x:
            return i #sortie anticipÃ©e (plus rapide qu'une boucle complÃ¨te)
    return False
```

Q C

```
#QC.(a)
def calcule(u0,f,n):
    u=u0
    S=u0
    for i in range(n):
        if u<0:   #on teste si f(u) est bien dÃ©fini
            return False
        u=f(u)
        S=S+u   #ne pas se tromper dans l'ordre des opÃ©rations
    return S

#QC.(b)
def f(x):
    return x**2+1
calcule(2019,f,1000)
```

Q D

```
#QD.(a)
def ecart(T):
    S=0
    n=len(T)
    for i in range(n):
        for j in range(n):
            S=S+(T[i][j]-T[0][0])**2
    return S

#QD.(b)
def suivant(T):
    n=len(T)
```

```
    #calcul de la somme des coefficients
    S=0
    for i in range(n):
        for j in range(n):
            S=S+T[i][j]
    #calcul des nouvelles valeurs des T[i][j] (calculs
    indÃ©pendants les uns des autres)
    N=n**2-1 #nombre de coefficients Ã  moyenner
    for i in range(n):
        for j in range(n):
            T[i][j]=(S-T[i][j])/N

#QD.(c)
def limite(T,seuil):
    nb=0 #compteur du nombre d'itÃ©rations
    while ecart(T)>seuil:
        suivant(T)
        nb=nb+1
    return [T[0][0],nb]
```

Solution du problème (X-ENS, MP, PC, PSI 2019 — sans la fin)

Q1 On fait attention à bien créer une nouvelle colonne à chaque itération.

```
def creerGrille(largeur, hauteur):
    return  [["VIDE for j in range(hauteur)] for i in range(largeur
    )]

#ou
def creerGrille(largeur, hauteur):
    Grille=[]
    for i in range(largeur):
        colonne=[]
        for j in range(hauteur):
            colonne.append("VIDE")
        Grille.append(colonne)
    return Grille
```

Q2

```
def ligne2chaine(grille,i):
    largeur=len(grille)
    res=""
    for j in range(largeur):
        if grille[i][j]=="VIDE":
            res=res+" "
        else:
             res=res+grille[i][j]
    return res

def afficheGrille(grille):
    hauteur=len(grille[0])
    for i in range(hauteur-1,-1,-1):
```

```
136         print(ligne2chaine(grille,i)
137
138 #ou
139 def afficheGrille(grille):
140     hauteur=len(grille[0])
141     for i in range(hauteur-1):
142         print(ligne2chaine(grille,hauteur-1-i))
```

Q3 On teste les k cases du haut de chaque colonne. On renvoie `True` dès qu'une colonne convient. Si aucune ne convient, on renvoie `False`.

```
146 def grilleLibre(grille,k):
147     largeur=len(grille)
148     hauteur=len(grille[0])
149     for x in range(largeur):
150         y=hauteur-k
151         while y<hauteur and grille[x][y]=="VIDE":
152             y+=1
153         if y==hauteur:
154             return True
155     return False
156
157 #ou
158 def grilleLibre(grille,k):
159     largeur=len(grille)
160     hauteur=len(grille[0])
161     for x in range(largeur):
162         # teste si la colonne x est libre
163         # avec une fonction auxiliaire pour pouvoir
164         # utiliser une boucle interrompue
165         def placedispo():
166             for y in range(hauteur-k, hauteur):
167                 if grille[x][y]!=VIDE:
168                     return False
169             return True
170         if placedispo():
171             return True
172     return False
```

La boucle interne comporte au maximum k itérations. La boucle externe comporte au maximum `largeur` itérations.

Chaque boucle s'effectue en temps constant. Ainsi $\boxed{\text{la complexité est en } O(k \times \texttt{largeur}).}$

Q4 Le barreau ne peut descendre que s'il n'est pas déjà en bas de l'aire de jeu et si la case sous le barreau est vide. Dans ce cas on fait descendre le barreau case par case en commençant par le bas (pour ne pas écraser de case non vide).

```
175 def descente(grille,x,y,k):
176     if y!=0 and grille[x][y-1]=="VIDE":
177         for j in range(y,y+k):
178             grille[x][j-1]=grille[x][j]
179         grille[x][y+k-1]="VIDE"
```

Q5 On commence par tester la possibilité de déplacement par rapport aux bords de la grille puis on vérifie si les cases d'arrivée sont vides. Une fois ces vérifications effectuées on effectue le déplacement

```
182 def deplacerBarreau(grille,x,y,k,direction):
183     largeur=len(grille)
184     if 0>x+direction or x+direction >= largeur:
185         return #interruption sans rien renvoyer
186     for j in range(y,y+k):
187         if grille[x+direction][j]!="VIDE":
188             return #interruption sans rien renvoyer
189     for j in range(y,y+k):
190          grille[x+direction][j]=grille[x][j]
191          grille[x][j]="VIDE"
192 
193 #ou
194 
195 def deplacerBarreau(grille,x,y,k,direction):
196     largeur=len(grille)
197     if 0<=x+direction and x+direction < largeur: #test bords
198         j=y
199         while j<y+k and grille[x+direction][j]=="VIDE":
200             j+=1
201         if j == y+k: # la destination est complÃ¨tement vide
202             for j in range(y,y+k):
203                 grille[x+direction][j]=grille[x][j]
204                 grille[x][j]="VIDE"
```

Q6 On stocke la case du haut dans une variable tampon et on déplace de haut en bas (pour ne pas écraser de case).

```
207 def permuterBarreau(grille,x,y,k):
208     tampon=grille[x][y+k-1]
209     for j in range(y+k-1,y,-1):
210         grille[x][j]=grille[x][j-1]
211     grille[x][y]=tampon
```

Q7 On peut être tenté (épreuve en temps limité) de réitérer `descente` tant que c'est possible. C'est pour cela que le sujet donne la contrainte sur la complexité.

La boucle conditionnelle (`while`), permet de trouver le pas de la descente et a une complexité en O(`hauteur`).

Puis la boucle inconditionnelle (`for`) qui suit a une complexité en $O(k)$.

```
214 def descenteRapide(grille,x,y,k):
215     pas=0
216     while y>=pas+1 and grille[x][y-pas-1]=="VIDE":
217         pas+=1
218     if pas !=0:
219         for j in range(k):
220             grille[x][y+j-pas]=grille[x][y+j]
221             grille[x][y+j]="VIDE"
```

Q8 On obtient les trois configurations successives suivantes :

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

				N	
				N	J
	B		**B**	**N**	N
N	N	**B**	J	**V**	V
N	**B**	J	**B**	**V**	V
V	N	J	B	**V**	J

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

Ce qui donne $(2+2)+(1+1+1+1)=\boxed{8}$ points.

Q9 On parcourt `rangee` en une fois. On note la couleur précédente (ou éventuellement `"VIDE"`) dans la variable `couleur`. Si la couleur est identique à la précédente, on incrémente la variable `m`.

Si la couleur est distincte mais que `m`≥ 3, on met à jour la variable entière `score` et le tableau `marking`.

En fin de boucle, on traite l'éventuel dernière répétition de couleurs.

```
224 def detecteAlignement(rangee):
225     n=len(rangee)
226     marking=[False for j in range(n)]
227     score=0
228     couleur=rangee[0]
229     m=1
230     for i in range(1,n):
231         if rangee[i]=="VIDE" or rangee[i]!=couleur:
232             if m>=3:
233                 score+=m-2
234                 for j in range(m):
235                     marking[i-1-j]=True
236             m=1
237             couleur=rangee[i]
238         else:
239             m+=1
240     if m>=3:
241         score+=m-2
242         for j in range(m):
243             marking[n-1-j]=True
244     return [marking, score]
```

Q10 (a) On parcourt toutes les rangées possibles extraites de la grille dans toutes les directions possibles. On remarque que la direction `(dx,dy)` dirige les mêmes rangées que la direction `(-dx,-dy)` (en partant de l'extrémité opposée). Ainsi toutes les rangées possibles sont données, sans répétition, par :

- rangées horizontaux, partant du bord gauche :
  `(i,j,dx,dy)=(0,j,1,0)` avec $0 \leq$ `j` $<$ *hauteur*.
- rangées verticaux, partant du bord inférieur :
  `(i,j,dx,dy)=(i,0,0,1)` avec $0 \leq$ `i` $<$ *largeur*.
- rangées diagonales montant vers la droite, partant du bord inférieur :
  `(i,j,dx,dy)=(0,j,1,1)` avec $0 \leq$ `j` $<$ *hauteur*.
- rangées diagonales montant vers la droite, partant du bord gauche :
  `(i,j,dx,dy)=(i,0,1,1)` avec $1 \leq$ `i` $<$ *largeur* (on commence à $i=1$ car $i=0$ a été

traité ci-dessus).

- rangées diagonales descendant vers la droite, partant du bord supérieur : `(i,j,dx,dy)=(i,hauteur-1,1,-1)` avec $0 \leq$ `i` $<$ *largeur*.
- rangées diagonales descendant vers la droite, partant du bord gauche : `(i,j,dx,dy)=(0,j,1,-1)` avec $0 \leq$ `j` $<$ *hauteur* -1 (on finit à `hauteur-2` car `hauteur-1` a été traité ci-dessus).

Au total il y a `3*(hauteur+largeur)-2` rangées à parcourir, soit sur l'exemple $\boxed{46 \text{ rangées.}}$

Remarque : certaines diagonales (il y en a 6) sont de longueur strictement inférieur à 3 et pourraient être supprimées de la liste .

(b) Il suffit d'appliquer les consignes.

```
247 def scoreRangee(grille, g, i, j, dx, dy):
248     largeur=len(grille)
249     hauteur=len(grille[0])
250     x,y=i,j
251     rangee=[]
252     while 0<=x and x<largeur and 0<=y and y<hauteur:
253         rangee.append(grille[x][y])
254         x+=dx
255         y+=dy
256     [marking,score]=detecteAlignement(rangee)
257     n=len(rangee)
258     for p in range(n):
259         if marking[p]:
260             g[i+p*dx][j+p*dy]="VIDE"
261     return score
```

Q11

```
264 def effaceAlignement(grille):
265     largeur=len(grille)
266     hauteur=len(grille[0])
267     score=0
268     #VRAIE copie de la grille :
269     g=[[grille[i][j] for j in range(hauteur)] for i in range(
    largeur)]
270     #test des lignes
271     dx,dy=1,0
272     for j in range(hauteur):
273         score+= scoreRangee(grille, g, 0, j, dx, dy)
274     #test des colonnes
275     dx,dy=0,1
276     for i in range(largeur):
277         score+= scoreRangee(grille, g, i, 0, dx, dy)
278     #test des diagonales montant vers la droite
279     dx,dy=1,1
280     for i in range(largeur):
281         score+= scoreRangee(grille, g, i, 0, dx, dy)
282     for j in range(1,hauteur):
283         score+= scoreRangee(grille, g, 0, j, dx, dy)
284     #test des diagonales descendant vers la droite
285     dx,dy=1,-1
286     for i in range(largeur):
287         score+= scoreRangee(grille, g, i, hauteur-1, dx, dy)
```

```
288         for j in range(hauteur-1):
289             score+= scoreRangee(grille, g, 0, j, dx, dy)
290         #renvoi du rÃ©sultat
291         return [g,score]
```

La copie de grille a une complexité en O(`hauteur*largeur`)

La fonction `scoreRangee` si on itère en `largeur` (respectivement en `hauteur`) a une complexité en

O(`hauteur+largeur`) (respectivement en O(`largeur`))

donc chacune de ces boucles a une complexité en O(`hauteur*largeur`)

En conclusion, par somme, la complexité de la fonction `effaceAlignement` est en $\boxed{O(\texttt{hauteur*largeur})}$.

Q12 Sur chaque colonne, chaque case doit descendre du nombre de cases vides situées sous elle. Pour ne pas écraser de case, on parcourt chaque colonne de bas en haut en comptant le nombre `k` de cases vides déjà rencontrées et en faisant descendre chaque case non vide rencontrée de `k` cases.

```
294 def tassementGrille(grille):
295     largeur=len(grille)
296     hauteur=len(grille[0])
297     for x in range(largeur):
298         k=0
299         for y in range(hauteur):
300             if grille[x][y]=="VIDE":
301                 k=k+1
302             elif k>=1:
303                grille[x][y-k]=grille[x][y]
304                grille[x][y]="VIDE"
```

Q13 On effectue une suite d'effacements suivis de tassements, en sommant les points obtenus jusqu'à ce qu'il n'y ait plus d'évolution (quand le score n'évolue pas).

```
307 def calculeScore(grille):
308     largeur=len(grille)
309     hauteur=len(grille[0])
310     scoreTotal=0
311     score=1
312     while score!=0:
313         [grille,score]= effaceAlignement(grille)
314         tassementGrille(grille)
315         scoreTotal=scoreTotal+score
316     return scoreTotal
```