Algorithme de Jeu de Scrabble

TIPE 2017-2018

Vincent Guzniczak

Sommaire

Problématique : Nous avons cherché à créer un algorithme de Scrabble dans nos hypothèses de travail fixées.

Plan

I°/ Algorithmes de mise en place du jeu

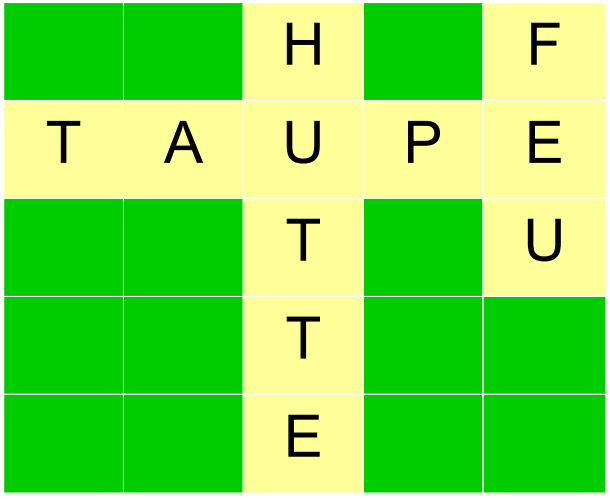
II°/ Recherche et travail sur les cases candidates

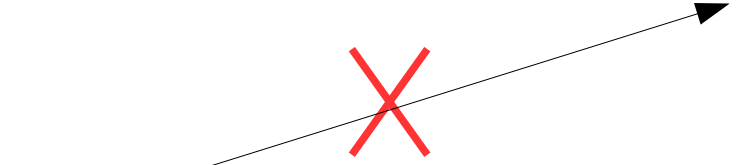
III°/ Recherche de mots solutions

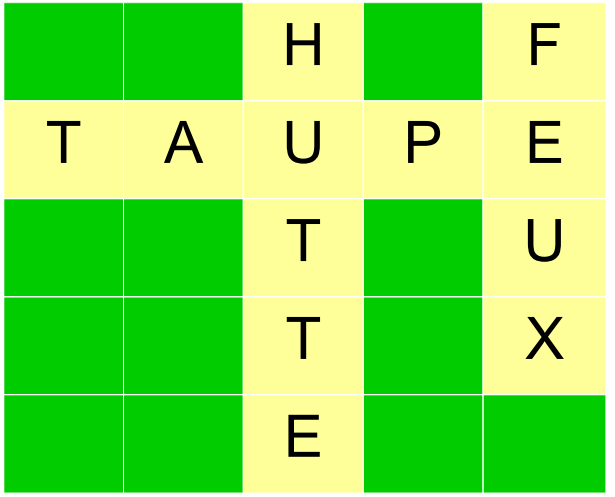
IV°/ Optimisation de l'algorithme final

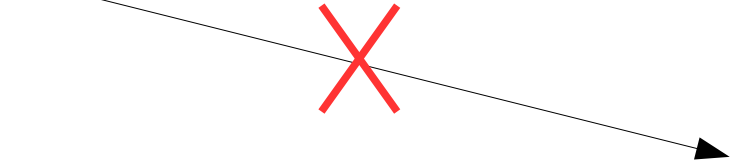
Introduction : Les hypothèses de notre travail

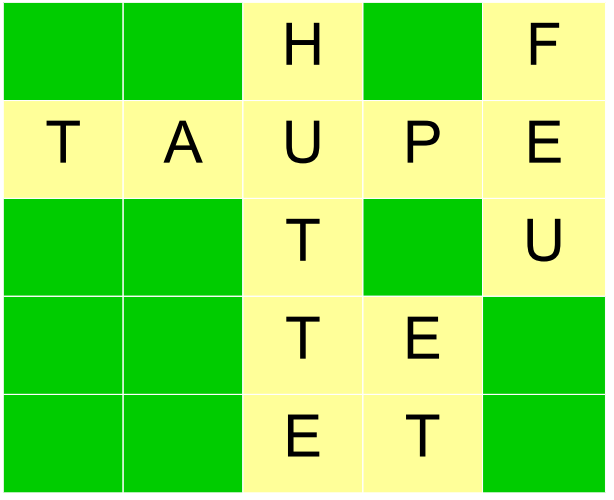
(a) On ne peut pas « compléter un mot » déjà en place en y ajoutant des lettres : par exemple un 's' ou un 'x' à la fin d'un mot pour obtenir un pluriel











(b) On ne peut pas placer de « doubles mots » : on ne peut pas placer de lettres qui créent plusieurs mots à la fois

I°/ Mise en place du jeu

1°/ Modèle Matriciel

(a) Matrice n°1 : **Array**

C'est la matrice de travail, que l'on utilise pour le calcul des points des mots placés à chaque tour.

"41" est "Lettre compte double"
"42" est "Lettre compte triple"
"43" est "Mot compte double"
"44" est "Mot compte triple"
"49" est la "Case départ
" 0 " est une case vide

```
>>> Mat
array([[44,  0,  0, 41,  0,  0,  0, 43,  0,  0,  0, 41,  0,  0, 44],
       [ 0, 43,  0,  0,  0, 42,  0,  0,  0, 42,  0,  0,  0, 43,  0],
       [ 0,  0, 43,  0,  0,  0, 41,  0, 41,  0,  0,  0, 43,  0,  0],
       [41,  0,  0, 42,  0,  0,  0, 41,  0,  0,  0, 43,  0,  0, 41],
       [ 0,  0,  0,  0, 43,  0,  0,  0,  0,  0, 43,  0,  0,  0,  0],
       [ 0, 42,  0,  0,  0, 42,  0,  0,  0, 42,  0,  0,  0, 42,  0],
       [ 0,  0, 41,  0,  0,  0, 41,  0, 41,  0,  0,  0, 41,  0,  0],
       [44,  0,  0, 41,  0,  0,  0, 49,  0,  0,  0, 41,  0,  0, 44],
       [ 0,  0, 41,  0,  0,  0, 41,  0, 41,  0,  0,  0, 41,  0,  0],
       [ 0, 42,  0,  0,  0, 42,  0,  0,  0, 42,  0,  0,  0, 42,  0],
       [ 0,  0,  0,  0, 43,  0,  0,  0,  0,  0, 43,  0,  0,  0,  0],
       [41,  0,  0, 43,  0,  0,  0, 41,  0,  0,  0, 43,  0,  0, 41],
       [ 0,  0, 43,  0,  0,  0, 41,  0, 41,  0,  0,  0, 43,  0,  0],
       [ 0, 43,  0,  0,  0, 42,  0,  0,  0, 42,  0,  0,  0, 43,  0],
       [44,  0,  0, 41,  0,  0,  0, 44,  0,  0,  0, 41,  0,  0, 44]])
```

(b) Matrice n°2 : **Chararray**

C'est la matrice d'affichage, renvoyée à l'utilisateur, où le jeu est joué.

C'est une matrice de caractères.

```
>>> M
array([['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']],
      dtype='<U1')
```

I°/ Mise en place du jeu

2°/ Lettres et utilisation

(a) **Bases du jeu**

(i) **Création d'un « Sac »** : une liste de lettres avec tous les 102 jetons du Scrabble.

```
>>> Sac=['Joker','Joker','e','e','e','e','e','e','e','e'
,'e','e','e','e','e','e','e','a','a','a','a','a','a','a'
,'a','a','i','i','i','i','i','i','i','i','n','n','n','n'
,'n','n','o','o','o','o','o','o','r','r','r','r','r','r'
,'s','s','s','s','s','s','t','t','t','t','t','t','u','u'
,'u','u','u','u','l','l','l','l','l','d','d','d','m','m'
,'m','g','g','b','b','c','c','p','p','f','f','h','h','v'
,'v','j','q','k','w','x','y','z']
```

(ii) **Valeurs des lettres** :

-Valeur au Scrabble : c'est la valeur des lettres en points brut selon les règles du Scrabble.

```
>>> Valeur_Scrabble('e')
1
>>> Valeur_Scrabble('w')
10
>>> Valeur_Scrabble('d')
2
>>> Valeur_Scrabble('x')
10
>>> Valeur_Scrabble('a')
1
```

-Valeur de travail : on associe aux lettres les valeurs égales à leur rang dans l'alphabet.

La valeur de travail de chaque lettre composant chaque mot rentré dans la grille de Scrabble sera placée dans la matrice « Array » à son emplacement équivalent à celui dans la matrice « Chararray ».

```
>>> Valeur_travail('a')
1
>>> Valeur_travail('b')
2
>>> Valeur_travail('c')
3
.
.
.
>>> Valeur_travail('x')
24
>>> Valeur_travail('y')
25
>>> Valeur_travail('z')
26
```

I°/ Mise en place du jeu

2°/ Lettres et utilisation

(b) Utilisation des lettres

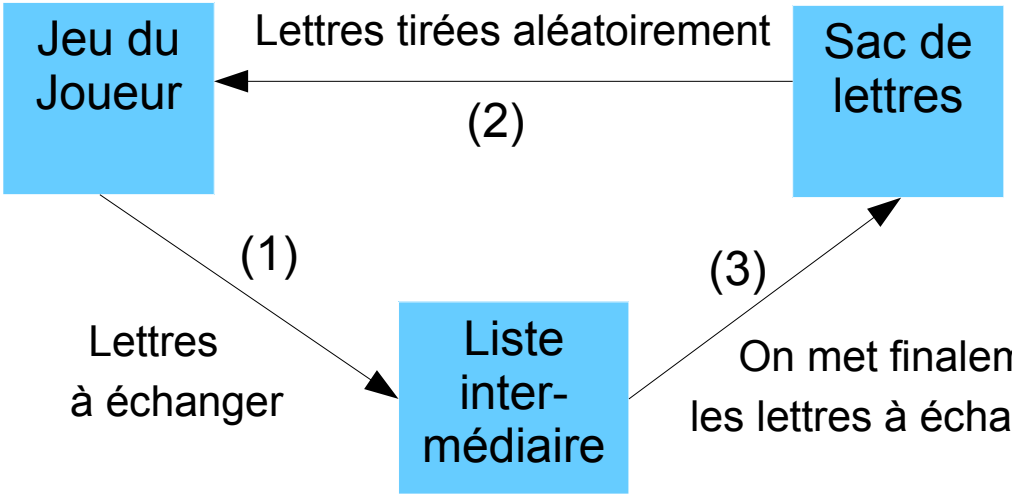
(i) Distribution initiale

Dans tous les algorithmes d'échanges/Distribution de lettres nous faisons usage du module « random » de Python.

```
>>> Distribution_initiale_de_lettres()
(['t', 'a', 'n', 'n', 'v', 'u', 'a'], ['h', 'r', 'm', 't', 'Joker', 'e', 'e'])

>>> Distribution_initiale_de_lettres()
(['b', 'i', 'c', 'p', 'k', 'a', 'a'], ['w', 'e', 'e', 'o', 'e', 't', 'm'])

>>> Distribution_initiale_de_lettres()
(['i', 't', 'i', 's', 'a', 'a', 'f'], ['c', 'l', 'f', 'i', 'i', 'l', 'd'])
```

(ii) Échange de lettres

Principe :




```
>>> échange_de_Lettres(['h','i','o','c','p','r','e'],[1,0,0,1,1,0,0])
['i', 'o', 'r', 'e', 'n', 'i', 'm']

>>> échange_de_Lettres(['h','i','o','c','p','r','e'],[1,1,1,1,0,0,1])
['p', 'r', 'a', 'e', 't', 'r', 'r']

>>> échange_de_Lettres(['h','i','o','c','p','r','e'],[0,1,0,1,0,0,0])
['h', 'o', 'p', 'r', 'e', 'e', 'd']
```

I°/ Mise en place du jeu

3°/ Remplissage de matrice et comptage de points

(a) Remplissage : Deux algorithmes similaires ; remplissage Haut-Bas, remplissage Gauche-Droite

L'utilisateur donne en argument un mot et les coordonnées i,j où l'on commence à entrer ce mot dans la matrice.

```
>>> Remplir_Matrice_GaucheDroite_avec_chararray('exemple',3,3)
24

>>> M
array([['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'e', 'x', 'e', 'm', 'p', 'l', 'e', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']],
      dtype='<U1')
```

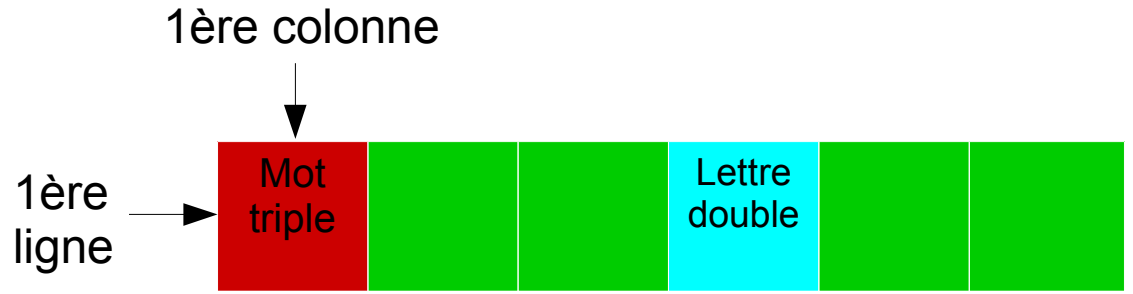
```
>>> Mat
array([[44,  0,  0, 41,  0,  0,  0, 43,  0,  0,  0, 41,  0,  0, 44],
       [ 0, 43,  0,  0,  0, 42,  0,  0,  0, 42,  0,  0,  0, 43,  0],
       [ 0,  0, 43,  0,  0,  0, 41,  0, 41,  0,  0,  0, 43,  0,  0],
       [41,  0,  0,  5, 24,  5, 13, 16, 12,  5,  0, 43,  0,  0, 41],
       [ 0,  0,  0,  0, 43,  0,  0,  0,  0,  0, 43,  0,  0,  0,  0],
       [ 0, 42,  0,  0,  0, 42,  0,  0,  0, 42,  0,  0,  0, 42,  0],
       [ 0,  0, 41,  0,  0,  0, 41,  0, 41,  0,  0,  0, 41,  0,  0],
       [44,  0,  0, 41,  0,  0,  0, 49,  0,  0,  0, 41,  0,  0, 44],
       [ 0,  0, 41,  0,  0,  0, 41,  0, 41,  0,  0,  0, 41,  0,  0],
       [ 0, 42,  0,  0,  0, 42,  0,  0,  0, 42,  0,  0,  0, 42,  0],
       [ 0,  0,  0,  0, 43,  0,  0,  0,  0,  0, 43,  0,  0,  0,  0],
       [41,  0,  0, 43,  0,  0,  0, 41,  0,  0,  0, 43,  0,  0, 41],
       [ 0,  0, 43,  0,  0,  0, 41,  0, 41,  0,  0,  0, 43,  0,  0],
       [ 0, 43,  0,  0,  0, 42,  0,  0,  0, 42,  0,  0,  0, 43,  0],
       [44,  0,  0, 41,  0,  0,  0, 44,  0,  0,  0, 41,  0,  0, 44]])
```

L'algorithme modifie la matrice Chararray **et** la matrice Array, et retourne le nombre de points gagnés.

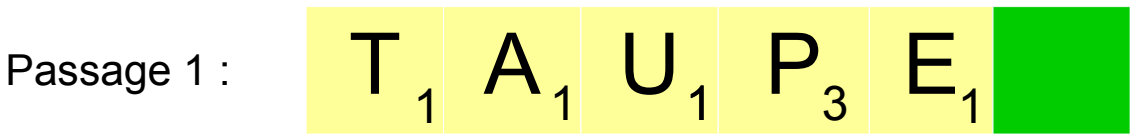
I°/ Mise en place du jeu

3°/ Remplissage de matrice et comptage de points

(b) **Comptage de points :** Le comptage se fait en parcourant deux fois la zone (i,j,len('mot'))

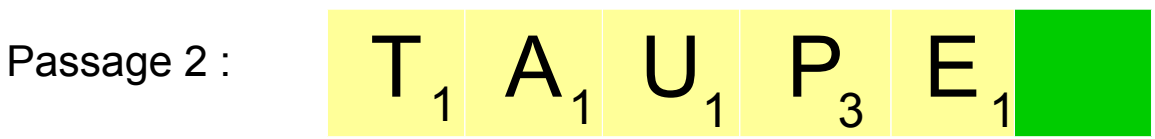


(i) On vérifie au premier passage grâce à la matrice Array la présence de cases « lettres compte double/triples » et on calcul la valeur du mot associée.



Valeur = 1 + 1 + 1 + 3x2 + 1 = 10

(ii) On vérifie au deuxième passage la présence de cases mot compte double/triple, et éventuellement on multiplie la valeur du mot précédente.



Valeur Finale = Valeur x 3 = 30

II°/Recherche et travail sur les cases candidates

1°)Vérification des hypothèses gardées

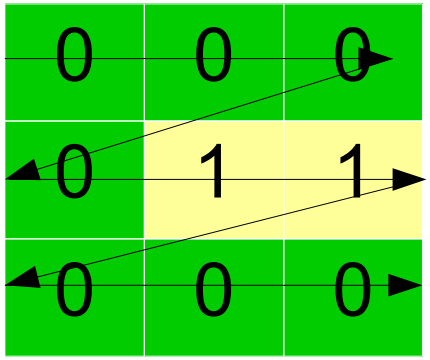
Préliminaire ; Recherche de cases candidates

On parcourt la grille et on repère toutes les cases non-vides ainsi que leur coordonnées i et j (en travaillant sur la matrice Array).

```
>>> Remplir_Matrice_HautBas_avec_chararray('exemples',1,3)
23

>>> M
array([['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'e', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'x', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'e', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'm', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'p', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'l', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'e', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 's', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']],
      dtype='<U1')

>>> Recherche_cases_candidates()
[[1, 3, 'e'], [2, 3, 'x'], [3, 3, 'e'], [4, 3, 'm'], [5, 3, 'p'], [6, 3, 'l'], [7, 3, 'e'], [7, 7, ''], [8, 3, 's']]
```

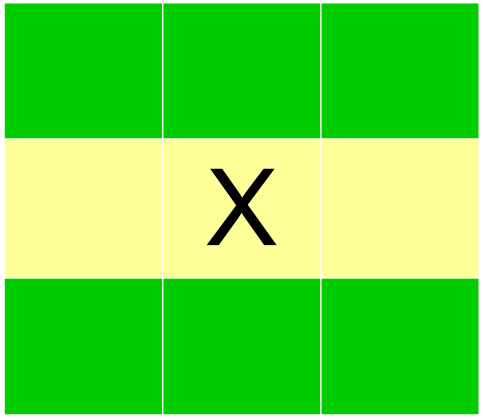
II°/Recherche et travail sur les cases candidates

1°)Vérification des hypothèses de travail : listes d'entourage



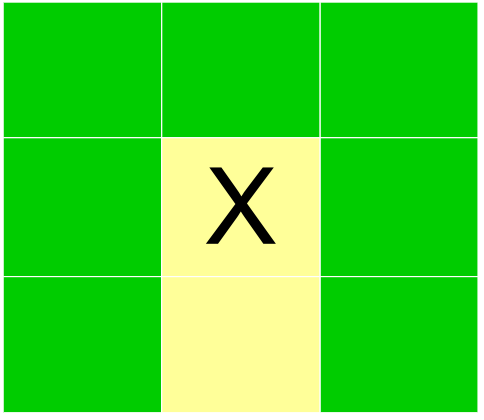
Une fois la lettre X repérée, on crée une **« Liste d'entourage » L** selon les cases l'entourant, remplie de 0 (case vide) et de 1 (case pleine) pour savoir si la case est disponible et, cas échéant, si on peut écrire dessus de haut en bas ou de droite à gauche.

Ici, nous sommes dans le cas « normal » ; la case candidate est « dans » la planche, pas sur un bord ni sur un coin.



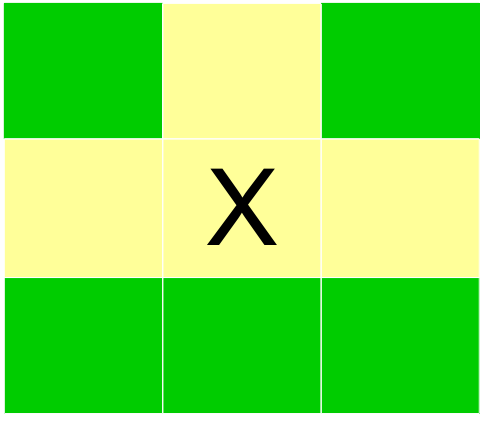
L = [0,0,0,1,1,1,0,0,0]

```
>>> Liste_Entourage_Normal(2,4)
[0, 0, 0, 1, 1, 1, 0, 0, 0]
```



L = [0,0,0,0,1,0,0,1,0]

```
>>> Liste_Entourage_Normal(10,3)
[0, 0, 0, 0, 1, 0, 0, 1, 0]
```



L = [0,1,0,1,1,1,0,0,0]

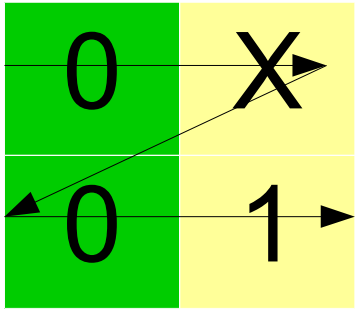
```
>>> Liste_Entourage_Normal(2,3)
[0, 1, 0, 1, 1, 1, 0, 0, 0]
```

II°/Recherche et travail sur les cases candidates

1°)Vérification des hypothèses de travail : listes d'entourage

Si la case se trouve sur un bord ou sur un coin de la grille, nous appliquons un raisonnement identique sur des « listes entourage » adaptées :

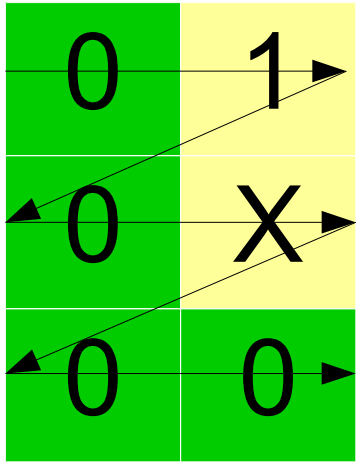
Sur un coin ;



L = [0,1,0,1]

```
>>> Liste_Entourage_Coin(0,14)
[0, 1, 0, 1]
```

Sur un bord ;



L = [0,1,0,1,0,0]

```
>>> Liste_Entourage_Bord(4,14)
[0, 1, 0, 1, 0, 0]
```

II°/Recherche et travail sur les cases candidates

2°) Vérification Haut-Bas

Une fois que l'on a confirmé pour une case que les hypothèses de travail sont vérifiées, on détermine, en utilisant à nouveau la valeur de la liste d'entourage associée, si l'on peut y écrire de Gauche à Droite ou de Haut en Bas.

(E1) (E2) (E3)

```
>>> M
array([['e', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['x', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['e', 'x', 'e', 'm', 'p', 'l', 'e', 's', '', '', '', '', '', '', ''],
       ['m', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['p', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['l', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['e', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['s', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']],
      dtype='<U1')
```

(E4)

Le tout est finalement incorporé dans un algorithme, renvoyant la donnée K :

#K = 0 si l'on ne peut pas y écrire
#K = 1 si l'on peut y écrire verticalement
#K = 2 si l'on peut y écrire horizontalement

(E1)
```
>>> Hypothèses_valides_et_Vérif_Haut_Droite(0,0)
2
```

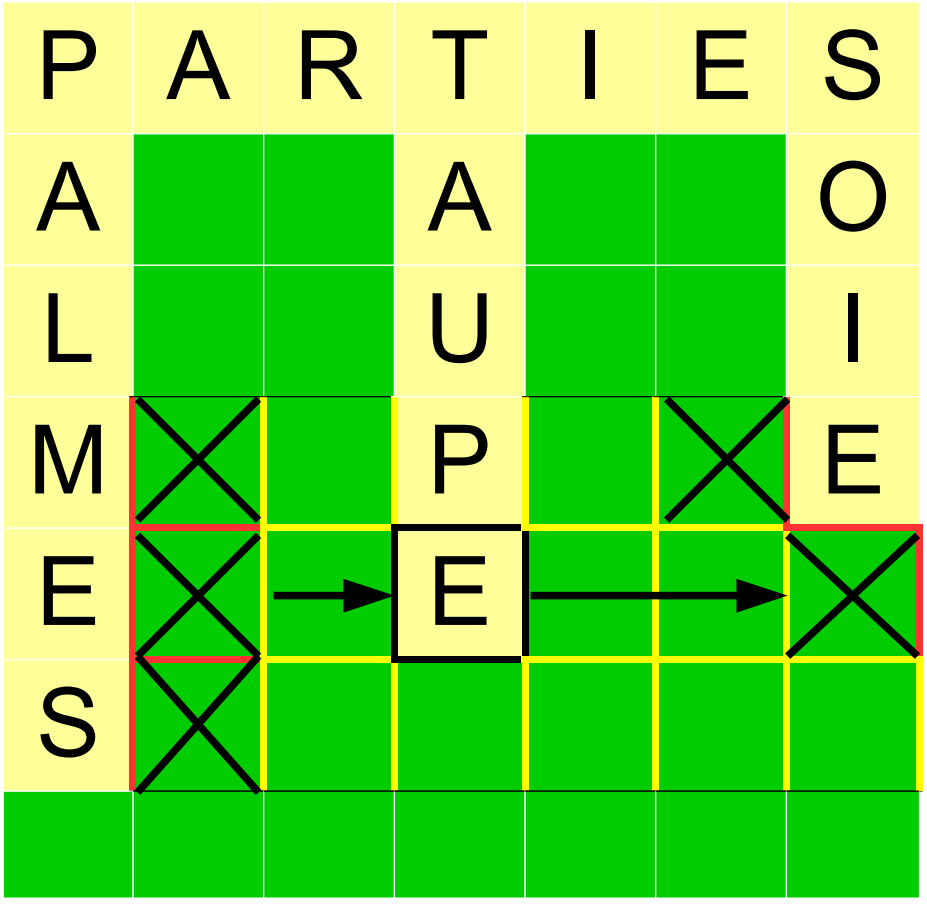
(E2)
```
>>> Hypothèses_valides_et_Vérif_Haut_Droite(2,0)
0
```

(E3)
```
>>> Hypothèses_valides_et_Vérif_Haut_Droite(2,2)
1
```

(E4)
```
>>> Hypothèses_valides_et_Vérif_Haut_Droite(6,0)
2
```

II°/Recherche et travail sur les cases candidates

3°) Cylindre de sécurité



Une fois que l'on sait si l'on peut écrire de Gauche à Droite / de Haut en Bas sur une case candidate, on créée un « cylindre de sécurité » sur les trois lignes/colonnes adjacentes à la case.

Selon la longueur minimum de chaque ligne/colonne de celui-ci, on détermine sur combien de cases on peut écrire à droite et à gauche de la lettre dans nos hypothèses de travail.

Exemple ; ici, pour la case E, on peut écrire sur 2 cases à droite et sur 1 case à gauche de la lettre.

II°/Recherche et travail sur les cases candidates

3°) Cylindre de sécurité

(E2) (E3) (E1)

```
>>> M
array([['p', 'a', 'r', 't', 'i', 'e', 's', '', '', '', '', '', '', '', ''],
       ['a', '', '', 'a', '', '', 'o', '', '', '', '', '', '', '', ''],
       ['l', '', '', 'u', '', '', 'i', '', '', '', '', '', '', '', ''],
       ['m', '', '', 'p', '', '', 'e', '', '', '', '', '', '', '', ''],
       ['e', '', '', 'e', '', '', '', '', '', '', '', '', '', '', ''],
       ['s', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']],
      dtype='<U1')
```

L'algorithme renvoie alors une liste de 3 éléments :
Le nombre de places pour écrire à Gauche/ en Haut, le nombre de places pour écrire à droite/ en bas, et la valeur de K (on l'utilise encore plus tard).

(E1)
```
>>> Cylindre_de_sécurité(4,3)
[1, 2, 2]
```

(E2)
```
>>> Cylindre_de_sécurité(0,6)
[0, 0, 0]
```

(E3)
```
>>> Cylindre_de_sécurité(2,6)
[1, 8, 2]
```

III°/Recherche de mots solutions

1°) Création d'une base de données adaptée

(a) Source : Dictionnaire FREELANG Bernard Vivier (22 740 mots)

(b) Problème : Les lettres du Scrabble sont toutes non-accentuées, contrairement à celles du dictionnaire, et Python ne reconnaît pas une lettre avec et sans accent de la même manière !

Exemple :

```
def Existence(mot) :
    L = ['élève','Scrabble', 'informatique']
    return(mot in L)

>>> Existence('eleve')
False
>>> Existence('élève')
True
```

Le mot 'eleve' n'est pas dans la liste bien que le mot 'élève' s'y trouve ; il faut donc adapter notre dictionnaire au manque d'accent au Scrabble.

Pour ça nous créons une nouvelle base de donnée sans accent à l'aide d'algorithmes adaptés.

III°/Recherche de mots solutions

1°) Création d'une base de données adaptée : illustration

```
>>> Sans_Accent('élèves')
'eleves'

>>> Sans_Accent('âpre')
'apre'

>>> Sans_Accent('bordé')
'borde'

>>> Sans_Accent('parfait')
'parfait'

>>> Sans_Accent('être')
'etre'

>>> Sans_Accent('noël')
'noel'
```

```
def Nouveau_dictionnaire() :
    Nouveau_dictionnaire = open("dictionnaire","r")
    lignes = []
    L = []
    Q = []
    V = []
    with open("dictionnaire","r") as Nouveau_dictionnaire :
        for ligne in Test :
            lignes.append(ligne)
    n = len(lignes)
    for k in range(0,n) :
        L = list(lignes[k])
        del(L[-1])
        Q.append(''.join(L))

    for i in range (0,n) :
        if len(list(Q[i]))< 7 and len(list(Q[i]))>3:
            V.append (Q[i])
    m = len(V)
    P = []
    for j in range(0,m) :
        P.append(Sans_Accent(V[k]))
    return(P)
```

III°/Recherche de mots solutions

2°) Mots solutions (a) Utilisation d'itertools

Pour chaque case candidate « valide », on vérifie quels mots peut-y entrer l'ordinateur.

Pour cela, on place chaque lettre en question dans son jeu à l'aide des algorithmes précédents et on vérifie quels mots il peut créer à l'aide du module Python « Itertools ».

```
def Mots_Candidats(liste_lettre,k):
    s= liste_lettre
    return(list(itertools.permutations(s,k)))
```

L'algorithme nous renvoie une liste d'uplets des arrangements de lettres possibles.

```
>>> Mots_Candidats(['a','i','g'],2)
[('a', 'i'), ('a', 'g'), ('i', 'a'), ('i', 'g'), ('g', 'a'), ('g', 'i')]
```

Arrangements de 2 lettres parmi 3

```
>>> Mots_Candidats(['a','i','g'],3)
[('a', 'i', 'g'), ('a', 'g', 'i'), ('i', 'a', 'g'), ('i', 'g', 'a'), ('g', 'a', 'i'), ('g', 'i', 'a')]
```

Arrangements de 3 lettres parmi 3

```
>>> Mots_Candidats(['a','i','g','v'],3)
[('a', 'i', 'g'), ('a', 'i', 'v'), ('a', 'g', 'i'), ('a', 'g', 'v'), ('a', 'v', 'i'), ('a', 'v', 'g'), ('i', 'a', 'g'), ('i
, 'a', 'v'), ('i', 'g', 'a'), ('i', 'g', 'v'), ('i', 'v', 'a'), ('i', 'v', 'g'), ('g', 'a', 'i'), ('g', 'a', 'v'), ('g', 'i
, 'a'), ('g', 'i', 'v'), ('g', 'v', 'a'), ('g', 'v', 'i'), ('v', 'a', 'i'), ('v', 'a', 'g'), ('v', 'i', 'a'), ('v', 'i', 'g
), ('v', 'g', 'a'), ('v', 'g', 'i')]
```

Arrangements de 3 lettres parmi 4

III°/Recherche de mots solutions

2°) Mots solutions

(b) Vérifications des mots candidats

On vérifie si les arrangements de lettres trouvés avec itertools donnent un mot existant dans notre dictionnaire ;

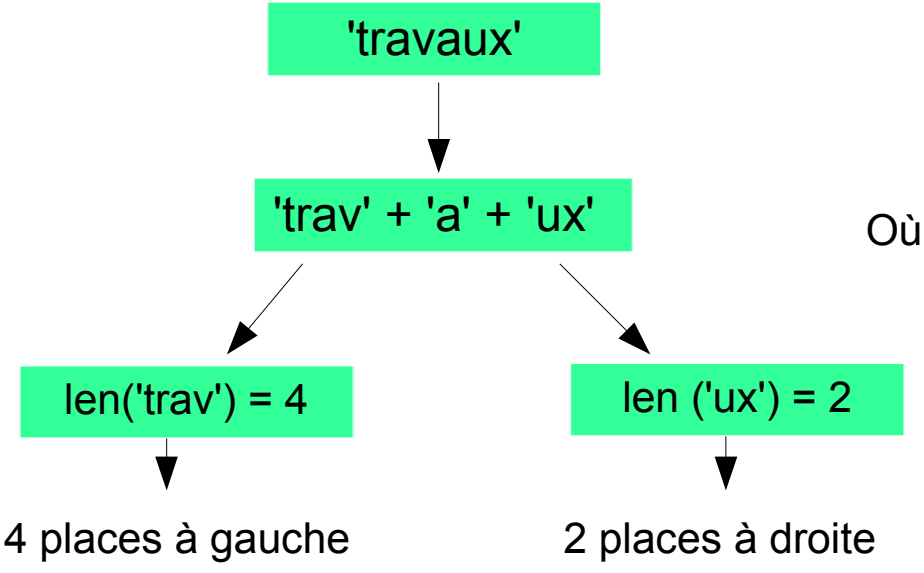
```
def Vérification_Mots_Candidats_Dictionnaire(liste_lettre,k) : # k est le nombre de lettres du mot "arrangé"

    v = len(Mots_Candidats(liste_lettre,k))
    Mots_Plaçables = []

    for i in range (0,v) :

        if (''.join(list(Mots_Candidats(liste_lettre,k)[i])) in Nouveau_Dictionnaire):

            Mots_Plaçables.append(''.join(list(Mots_Candidats(liste_lettre,k)[i])))

    return(Mots_Plaçables)    # Renvoie les mots existant dans notre dictionnaire
```

```
>>> Vérification_Mots_Candidats_Dictionnaire(['a','f','g','l','e','e','c','x'],3)
['age', 'ale', 'axe', 'fac', 'fee', 'gel', 'lac']
>>> Vérification_Mots_Candidats_Dictionnaire(['a','f','g','l','e','e','c','x'],4)
['agee', 'axee', 'face', 'fela', 'fele', 'gela', 'gele', 'lace', 'cafe', 'cage', 'cale', 'cela']
>>> Vérification_Mots_Candidats_Dictionnaire(['a','f','g','l','e','e','c','x'],5)
['fecal', 'galee', 'glace', 'lacee', 'calee']
>>> Vérification_Mots_Candidats_Dictionnaire(['a','f','g','l','e','e','c','x'],6)
['fecale', 'glacee']
```

III°/Recherche de mots solutions

3°) Placer les nouveaux mots

Nous vérifions tout d'abord si les mots justes précédents rentrent dans leur cylindre de sécurité en les cassant en deux autour de la case candidate avec un algorithme :



Où 'a' la lettre candidate

```
>>> Données_Mot_Juste('travaux','a')
(2, 4, 'tr', 'vaux')
```

```
>>> Données_Mot_Juste('travaux','v')
(3, 3, 'tra', 'aux')
```

```
>>> Données_Mot_Juste('travaux','u')
(5, 1, 'trava', 'x')
```

Notre algorithme prend en argument le mot et la lettre candidate, et renvoie le nombre de cases libre à gauche et à droite, puis la partie du mot à droite ainsi que celle à gauche de la lettre candidate initiale.

III°/Recherche de mots solutions

3°) Placer les nouveaux mots

Nous vérifions ensuite si les mots peuvent être placés autour sur la lettre candidate suivant les résultats de l'algorithme précédent et la taille du cylindre de sécurité.

```
def Vérif_Cylindre_Sécurité(mot,i,j):

    p = Données_Mot_Juste(mot,Mat[i][j])[0]
    q = Données_Mot_Juste(mot,Mat[i][j])[1]
    I = Cylindre_de_sécurité(i,j)[0]      # Place à Gauche
    J = Cylindre_de_sécurité(i,j)[1]      # Place à Droite

    return( (p<= I) and (q >= J))
```

Nous testons la valeur potentielle en points classons des mots résultats finaux, puis nous les classons avec un tri par insertion.

```
def Valeur_points_potentielle(mot,i,j,K):

    L = []
    n = len(mot)
    Pts_Mot = 0
    if K == 2 :  # Cas Haut Bas
        for k in range (0,n) :
            if Mat[i][j+k] == 41 :
                Pts_Mot = Pts_Mot + 2*Valeur_Scrabble(Mot_L[k])
            elif Mat[i][j+k] == 42:
                Pts_Mot = Pts_Mot + 3*Valeur_Scrabble(Mot_L[k])
            else:
                Pts_Mot = Pts_Mot + Valeur_Scrabble(Mot_L[k])
        for p in range(0,n) :
            if Mat[i][j+p] == 43 :
                Pts_Mot = 2*Pts_Mot
            elif Mat[i][j+p] == 44 :
                Pts_Mot = 3*Pts_Mot
        L.append(mot)
        L.append(i)
        L.append(j)
        L.append(Pts_Mot)
    # pas de place pour l'afficher, mais sinon K == 2 et on fait
    # la même chose de Haut en Bas en utilisant l'algo de remplissage
    return(L)
```

Un Tri par insertion nous donne alors, finalement, le meilleur mot à placer pour l'ordinateur.

IV°/Optimisation de l'algorithme final

1°) Résultats et problème

Avant optimisation de l'algorithme, nous avions un Dictionnaire de 129 000 mots (tiré d'une base de donnée affiliée au CNRS),et nous faisons une recherche de mots candidats de toutes les longueurs comprises entre 1 et 8.

```
>>> Joueur_Joue('eetginh',7,3)
array([['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'e', 'e', 't', 'g', 'i', 'n', 'h', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']],
      dtype='<U1')
```

Résultats : sur les premiers lancements test, le temps que met l'ordinateur à jouer son premier coup a été supérieur à 40 min !

```
>>> L_Ordinateur_Joue()
array([['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', 'h', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', 'o', '', '', '', '', '', '', '', '', ''],
       ['', '', '', 'e', 'e', 't', 'g', 'i', 'n', 'h', '', '', '', '', ''],
       ['', '', '', '', '', 'e', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', 'l', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
       ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']],
      dtype='<U1')
```

IV°/Optimisation de l'algorithme final

1°) Résultats et problème

Problème ; beaucoup trop grande complexité de l'algorithme, reposant surtout sur le nombre de Mots candidats d'itertools et de leur vérification dans le dictionnaire :

Notion mathématique d'« arrangement » :

$$A_n^k = n\,(n-1)\,(n-2)\cdots(n-k+1).$$

$$A_n^k = \frac{n!}{(n-k)!} \quad \text{pour } k \leq n,$$

$$\sum_{k=1}^{8} A_8^k = 109600$$

```
>>> len(Mots_Candidats(['e', 'e', 't', 'g', 'i', 'n', 'h','i'],8))
40320

>>> len(Mots_Candidats(['e', 'e', 't', 'g', 'i', 'n', 'h','i'],7))
40320

>>> len(Mots_Candidats(['e', 'e', 't', 'g', 'i', 'n', 'h','i'],6))
20160

>>> len(Mots_Candidats(['e', 'e', 't', 'g', 'i', 'n', 'h','i'],5))
6720

>>> len(Mots_Candidats(['e', 'e', 't', 'g', 'i', 'n', 'h','i'],4))
1680
```

Tenter de trouver tous les arrangements possibles et de vérifier la présence de tous les résultats dans la base de donnée est beaucoup trop calculatoire ; nous devons améliorer l'algorithme.

IV°/Optimisation de l'algorithme final

2°) Améliorations mises en place :

(a) Changement d'hypothèses :

(i) On ne cherche plus les arrangements de plus de 6 lettres, ni ceux de moins de 4 lettres.

(ii) On cherche d'abord les arrangements à 5 lettres. Si on trouve des mots plaçables, on s'arrête là. Sinon, on répète le procédé pour 4 lettres, éventuellement pour 6.

(b) Modification de la base de données :

(i) Autre base de donnée : (on choisit de travailler sur un dictionnaire plus court, le FREELANG de Bernard Vivier (22 740 mots)).

(ii) Suppression de mots inutiles ; suppression de tous les mots de plus de 8 lettres et de moins de 4 lettres pour se placer dans les nouvelles hypothèses :

```
for i in range (0,n) :
    if len(list(Q[i]))< 8 and len(list(Q[i]))>3:
        V.append (Q[i])
m = len(V)
```

(extrait de Nouveau_Dictionnaire)

(iii) Séparation de la base de données en 26 dans l'ordre alphabétique ; On réduit drastiquement le nombre de recherches en vérifiant la première lettre du mot que l'on cherche à valider.

Fin

* *

*