PUISSANCE 4

Logique du jeu :

-Menu
-Sélectionner une colonne
-Regarder si la colonne est valide (Pas remplie et dans le plateau)
-Attribuer pion à la case avec bonne couleur
-Passer au joueur suivant
-Checker si on a 4 pions alignés ou plateau rempli
Fonction **checkWin**

Affichage :

Afficher le plateau
Afficher les pions

Fonction Main :

 # On regarde si un des 2j a gagné
win = False
While not win :
…..
check_Win(win) --> return win

1ere solution : Créer une liste pour chaque colonnes, soit 7 listes avec 6 indices
Problème : vérifier si les pions sont alignés, cela prend beaucoup de lignes pour chaque liste
2ème solution : Créer un tableau en deux dimensions
Afficher sur le plateau, dans la bonne case l'emoji correspondant à la valeur stockée dans le tableau

VIDE = 0 (Afficher 2 espaces)
J1 (JAUNE) = 1 (Afficher un pion jaune)
J2 (ROUGE) = 2 (Afficher un pion rouge)

TABLEAU REPRÉSENTATIF DU STOCKAGE DES INFORMATIONS DU JEU :

Colonne 1	Colonne 2	Colonne 3	Colonne 4	Colonne 5	Colonne 6	Colonne 7
Case 6	Case 6	Case 6	Case 6	Case 6	Case 6	Case 6
Case 5	Case 5	Case 5	Case 5	Case 5	Case 5	Case 5
Case 4	Case 4	Case 4	Case 4	Case 4	Case 4	Case 4
Case 3	Case 3	Case 3	Case 3	Case 3	Case 3	Case 3
Case 2	Case 2	Case 2	Case 2	Case 2	Case 2	Case 2
Case 1	Case 1	Case 1	Case 1	Case 1	Case 1	Case 1

Comment créer et procéder à travers les fonctions avec ce type de stockage de données (grandes lignes) ?

-Sélectionner une colonne → modele.py
On demande à l'utilisateur d'entrer un numéro de colonne

-Regarder si la colonne est valide (Pas remplie et dans le plateau) → modele.py
On parcourt chaque indice de la deuxième dimension du tableau :
La colonne est accessible dans le plateau = la colonne sélectionnée appartient à un nombre compris dans la longueur du plateau (len)
La colonne est remplie = toutes les informations contenues dans les indices sont différentes de 0
La colonne n'est pas remplie = une des infos contenues dans les indices vaut 0

-Attribuer pion à la case avec la bonne couleur→ modele.py
On regarde dans la colonne choisie en parcourant chaque indice de de la colonne et en partant de l'indice 0 (case du bas)..
Lorsque qu'un indice est = 0 en partant du premier indice de la deuxième dim du tableau, il prend la valeur du joueur qui joue le coup("player_turn")

-Passer au joueur suivant → modele.py
Variable "player_turn" qui stocke la valeur du numéro du joueur
À la fin de chaque tour → on ajoute 1 à player_turn au modulo 2

-Checker si on a 4 pions alignés ou plateau rempli → modele.py
Plateau rempli = Tous les indices de toutes les colonnes sont != 0

Pions alignés verticalement = Pour chaque colonne :
if indice 0 & indice 1 & indice 2 & indice 3 == 1 ou 2 → **0 ou 1**
elif indice 1 & indice 2 & indice 3 & indice 4 == 1 ou 2
elif indice 2 & indice 3 & indice 4 & indice 5 == 1 ou 2

Pions alignés horizontalement = Pour chaque même indice du tableau dans chaque colonne :
if indice i col 1 & indice i col 2 & indice i col 3 & indice i col 4 == 1 or ...==2

Pions alignés en diagonale =

CHANGEMENT DES CONSTANTES → PB AVEC LE MODULO POUR LES TOURS DES JOUEURS
Nouvelles constantes :
VIDE = **-1** (Afficher 2 espaces)
J1 (JAUNE) = **0** (Afficher un pion jaune)
J2 (ROUGE) = **1** (Afficher un pion rouge)

FONCTION winCheck :

Cette dernière n'a **pas complètement été développée par nos soins, elle est fortement inspirée d'une autre fonction trouvée sur le net**. En effet après de multiples recherches et tests nous n'avons pas réussi à trouver une fonction viable et avons donc été contraints de faire des recherches internet dans le but de ne pas complètement bloquer sur le projet…

Nous avons assez patiemment essayé de trouver des solutions jusqu'au 3/01/20, ces solutions se rapprochaient de la fonction que nous avons pû trouver sur le net mais n'étaient tout de même pas à la hauteur…(Nous aurions dû les glisser dans ce fichier mais n'y avons pas pensé auparavant, nous nous en excusons)

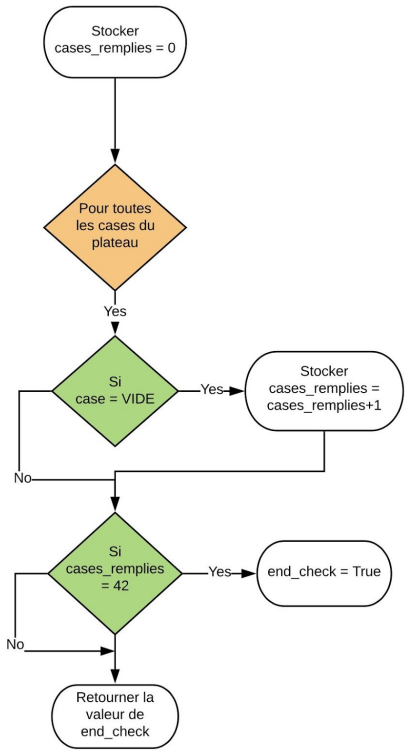
Lien du fichier : https://codes-sources.commentcamarche.net/source/101916-puissance4#

```
def line(b): #watch if we have 4 pion aligned in line
    for l in range(6):
        for k in range(4):
            if b[k][l] == b[k+1][l] == b[k+2][l] == b[k+3][l] == 1:
                return [1,b,k,l,k+1,l,k+2,l,k+3,l]
            if b[k][l] == b[k+1][l] == b[k+2][l] == b[k+3][l] == 2:
                return [2,b,k,l,k+1,l,k+2,l,k+3,l]
    return [0]

def column(b): #watch if we have 4 pion aligned in column
    for k in range(7):
        for l in range(3):
            if b[k][l] == b[k][l+1] == b[k][l+2] == b[k][l+3] == 1:
                return [1,b,k,l,k,l+1,k,l+2,k,l+3]
            if b[k][l] == b[k][l+1] == b[k][l+2] == b[k][l+3] == 2:
                return [2,b,k,l,k,l+1,k,l+2,k,l+3]
    return [0]

def diaglltr(b): #watch if we have 4 pion aligned in diagonal (downleft-upright)
    for k in range(4):
        for l in range(3):
            if b[k][l] == b[k+1][l+1] == b[k+2][l+2] == b[k+3][l+3] == 1:
                return [1,b,k,l,k+1,l+1,k+2,l+2,k+3,l+3]
            if b[k][l] == b[k+1][l+1] == b[k+2][l+2] == b[k+3][l+3] == 2:
                return [2,b,k,l,k+1,l+1,k+2,l+2,k+3,l+3]
    return [0]

def diagtllr(b): #watch if we have 4 pion aligned in diagonal (upleft-downright)
    for k in range(4):
        for l in range(3):
            if b[k][5-l] == b[k+1][4-l] == b[k+2][3-l] == b[k+3][2-l] == 1:
                return [1,b,k,5-l,k+1,4-l,k+2,3-l,k+3,2-l]
            if b[k][5-l] == b[k+1][4-l] == b[k+2][3-l] == b[k+3][2-l] == 2:
                return [2,b,k,5-l,k+1,4-l,k+2,3-l,k+3,2-l]
    return [0]
```

FONCTION TROUVÉE SUR LE NET

Ce document est un brouillon qui nous a servi au début pour lancer le projet et commencer à voir comment procéder.

Le projet étant terminé ou presque, il y avait de bonnes idées mais quelques unes étaient floues.



Cette fonction nous permet de vérifier si le plateau est rempli et donc d'arrêter la partie si nécessaire. On regarde toutes les cases du tableau à deux dimensions, si l'une d'entre elle n'est pas vide, on ajoute +1 à la variable **cases_remplies**. On compare ensuite cela aux nombre de cases total, si ces derniers correspondent cela signifie que le plateau est rempli. On renvoie alors une valeur "True" à la variable **end_check** dans la fonction main pour mettre fin à sa boucle Tant que, arrêter la partie et afficher qu'il y a égalité.

Le lucidchart que nous avions fait ne nous a en fait pas tellement servi, nous nous sommes en réalité servi de ce fichier pour créer le puissance 4. Une fois la version de base terminée, c'est le même mécanisme il nous suffisait alors de réfléchir un peu pour implémenter les autres modes de jeu.

Pour finir, nous n'avons pas réussi à résoudre le pb du mode defi +, en effet, lors de l'ajout d'un deuxième pion après l'utilisation d'un pion power, peu importe la colonne choisie, le pion n'est pas forcément assigné ds la bonne et ne correspond pas au bon emoji...