

ISN (Informatique et Sciences du Numérique)

Projet Othello

Enseignants : **M. CHAUVIN** et **M. GUERCH**

Personnes composant le groupe (3) :

- **COUCHOUD Thomas**
- **FROGER Olivier**
- **JACQUES Johann**

Dossier de **COUCHOUD Thomas**

Table des matières

I - Présentation du projet :	1
II - Répartitions des tâches :	2
III - Cahier des charges du projet :	2
IV - Présentation globale des différentes parties :	2
A - L'interface :	2
B - La communication :	3
C - L'algorithme de jeu :	3
V - Présentation détaillée de la partie « algorithme de jeu » :	3
A - Gestion des informations du plateau de jeu :	3
B - Gestion du placement des pions :	4
C - Retournement des pions :	6
VI) Détail des fonctions utilisées et fonctions d'informations :	6
A - GetColor (ligne 38) :	6
B - GetNumberColor (ligne 24):	6
C - DetectPawn (ligne 53) :	7
D - GetNextPawn (ligne 75) :	7
VI - Conclusion :	8
VII – Annexes.....	8
A – Game.py.....	8
B - Interface.py :	11

I - Présentation du projet :

L'enseignement de la « matière » ISN nous a permis de composer nous-même et, en partant de rien, un projet dans un langage de programmation défini (ici le [Python](#)).

Afin de répondre à cette demande, nous avons eu l'idée de reproduire le jeu [Othello](#), le rendant ainsi disponible depuis un ordinateur. Afin qu'il soit plus attrayant, nous avons pris la décision que notre programme opposerait deux joueurs humains (et pas un seul joueur qui se retrouve contre le PC), chacun sur deux ordinateurs différents. Les deux parties seraient ainsi synchronisées et chaque personne jouerait son tour au bon moment.

En ce qui concerne le jeu en lui-même, c'est un simple jeu de plateau basé sur les capacités

de réflexion des deux participants. L'othellier (plateau de jeu) est composé de soixante-huit cases (formé par un carré de côté de huit cases). La partie commence avec deux pions noirs et deux pions blancs au centre de l'aire de jeu de façon alternés. Le principe est d'« emprisonner » les pions de l'adversaire avec nos propres pions, les transformant ainsi en pions de notre couleur. La partie se termine quand les deux joueurs ne peuvent plus jouer. Le gagnant est celui qui possède le plus de pions.

II - Répartitions des tâches :

Notre groupe étant composé de trois personnes, nous avons pu définir trois tâches différentes pour chacun d'entre nous. Nous obtenons ainsi la répartition des tâches suivante :

- **COUCHOUD Thomas** : Création de l'algorithme déterminant les impacts des coups joués.
- **FROGER Olivier** : Création de l'interface utilisateur.
- **JACQUES Johann** : Création d'un système permettant la synchronisation entre les deux terminaux.

III - Cahier des charges du projet :

Le but à atteindre est le respect des règles du jeu concerné. Cela implique donc de disposer d'une interface représentant le plateau de jeu et de la règle de placement des pions. Le projet reposant sur une connexion entre deux ordinateurs, il nous faut donc un système capable de faire communiquer ceux-ci entre eux.

IV - Présentation globale des différentes parties :

A - L'interface :

La création de l'interface repose sur le module [TkInter](#) proposé par Python permettant la réalisation d'interfaces graphiques. Cette représentation graphique du jeu se compose de différentes parties :

- Une représentation du plateau de jeu. Celle-ci permet l'affichage des différents pions sur la grille du jeu. Elle inclut aussi la possibilité de cliquer dans une case pour placer l'un de ses pions lorsque c'est le tour du joueur en question.
- L'interface affiche des informations sur la partie en cours, notamment, le nombre de pions de chaque couleur présents sur la grille.
- Un "tchat" a été ajouté permettant de communiquer avec son adversaire.

B - La communication :

La communication est un point important dans notre projet. Il permet de synchroniser les deux ordinateurs entre eux (pions présents sur la grille, "tchat").

Ce processus se compose de deux parties :

- Un ordinateur « serveur » qui est uniquement capable de répondre à des demandes.
- Un ordinateur « client » qui envoie des demandes au serveur.

Les deux ordinateurs n'opérant pas de la même manière, il y a deux façons d'envoyer les informations :

- Lorsque le client doit envoyer une information au serveur, il le fait par message. Le serveur traitera l'opération et enverra une réponse comme quoi cette dernière a été prise en compte.
- A l'inverse, si c'est le serveur qui doit envoyer une information au client, cela est plus difficile. Le serveur n'est pas capable d'envoyer des messages, il peut seulement répondre à des demandes. Pour contourner le problème, nous opérons de la manière décrite ci-après. Le serveur stocke les informations à envoyer (positions d'un éventuel pion joué, message dans le "tchat") dans des variables afin de les retrouver plus tard. A un intervalle de temps défini, le client va envoyer une demande au serveur du type « quels sont les mises à jours qui se sont produites de ton côté ? ». A la réception de cette demande, le serveur va ainsi répondre en fonction de ce qui est stocké dans les variables citées précédemment, permettant ainsi au serveur de communiquer avec le client.

C - L'algorithme de jeu :

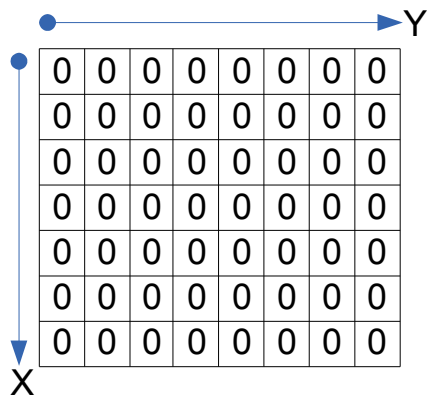
Cette partie a pour vocation de traiter tous les coups que l'interface aura récupéré de l'utilisateur. En fonction du placement du pion et de son environnement, plusieurs cas sont possibles :

- Le pion est dans une position impossible.
- Le pion ne peut être joué car il n' « emprisonne » aucun pion adverse.
- Le positionnement du pion est valide et il faut retourner tout les pions « emprisonnés ».

V - Présentation détaillée de la partie « algorithme de jeu » :

La partie algorithmique de jeu se trouve dans son propre fichier « [Game.py](#) » qui peut être importé par une autre partie du programme, dans notre cas l'interface graphique. Ce fichier permet de gérer une grille d'Othello de huit cases par huit comportant deux couleurs (couleur « 1 » et couleur « 2 »). Il contient aussi plusieurs fonctions permettant d'interagir avec cette grille (placer des pions) mais aussi d'obtenir des informations sur celle-ci (nombre de pions d'une couleur, couleur du pion à une position donnée).

A - Gestion des informations du plateau de jeu :



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Les informations du plateau sont, comme dit plus haut, conservées dans le fichier de l'algorithme de jeu et non pas par l'interface graphique. Le plateau de jeu est tout simplement représentée par un tableau à deux dimensions d'entiers contenant les couleurs des pions pour chaque position de la grille.

Exemple :

```
grille[x][y] = 5
```

L'assignation présentée au dessus positionnera ainsi la couleur « 5 » à la position (x;y).

Cette grille est créée lors de l'importation de ce fichier par l'interface ([ligne 164](#)), puis, grâce à la fonction « init », est initialisée. Cette fonction remplace toutes les valeurs du tableau par des « 0 » qui représentent une case vide, puis place les quatre pions de départ au centre du plateau de jeu. La fonction « init » est donc utilisée au premier lancement mais peut aussi être appelée pour recommencer une partie.

B - Gestion du placement des pions :

Maintenant que nous savons comment sont stockés les pions de la grille, il est intéressant d'étudier comment le programme gère le placement de nouveaux pions. Dans l'explication globale de cette partie du programme, nous avons vu plusieurs cas possibles, à savoir un pion placé dans une case invalide, un pion ne menant à aucun retournement ou bien une position valide.

Il est donc nécessaire de savoir dans quel cas nous nous trouvons. C'est ce que la fonction « place(color, x, y) » ([ligne 131](#)) est chargée de faire. Les paramètres nécessaires pour son fonctionnement sont :

- Une couleur représentée par un entier.
- La position « x » où est joué le pion.
- La position « y » où est joué le pion.

« place » va commencer par identifier si une position est « invalide » car si c'est le cas, le pion ne peut pas être placé et la fonction sera stoppée. Mais qu'est-ce qu'une position « invalide » ? On la retrouve notamment dans trois cas différents :

- La case demandée est hors de la grille.
- La case est déjà occupée par un autre pion.
- La position jouée n'as pas de pions de l'adversaire dans les huit cases l'entourant.
- La position jouée ne permet pas de retourner des pions adverses.

Isolons chaque cas.

1 - Commençons par celui où le pion est hors de la grille. Cette détection est simple et consiste juste en une vérification des coordonnées « x » et « y ». Si celles-ci ne sont pas comprises dans l'intervalle [0;7] (correspondant aux 8 lignes et 8 colonnes du tableau à deux

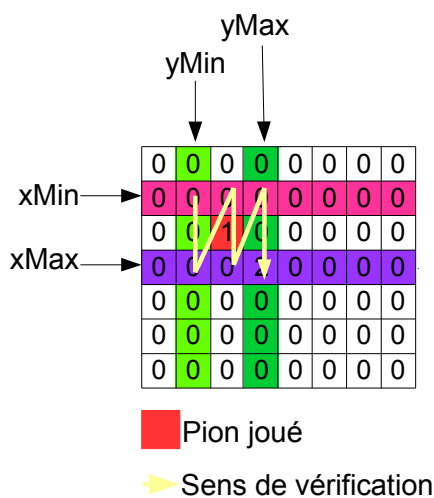
dimensions) c'est que le pion se trouve hors de la grille. En Python, cette vérification peut être effectuée par une instruction « if ». Dans l'illustration de la fonction réellement utilisée, le « 3 » renvoyé par la fonction représente le fait que le pion n'est pas dans la grille.

```
if(x < 0 or x > 7 or y < 0 or y > 7): return 3 #Hors de la grille
```

2 - Penchons-nous maintenant sur le cas où la case jouée est déjà occupée. Comme le cas précédent, un « if » permettra de remplir cette demande. Si la couleur de la position jouée n'est pas la couleur « vide », c'est que la case est déjà prise.

```
elif(getColor(x, y) != 0): return 1 #Si la case n'est pas vide, on ne peut pas jouer
```

3 - Autre cas possible, le pion joué n'a pas de pions adverses dans les cases qui l'entourent. Ce processus est réalisé par la fonction « hasPawnNext(color, x, y) » ([ligne 106](#)). Celle-ci va vérifier pour chaque case de son entourage si un pion adverse s'y trouve.



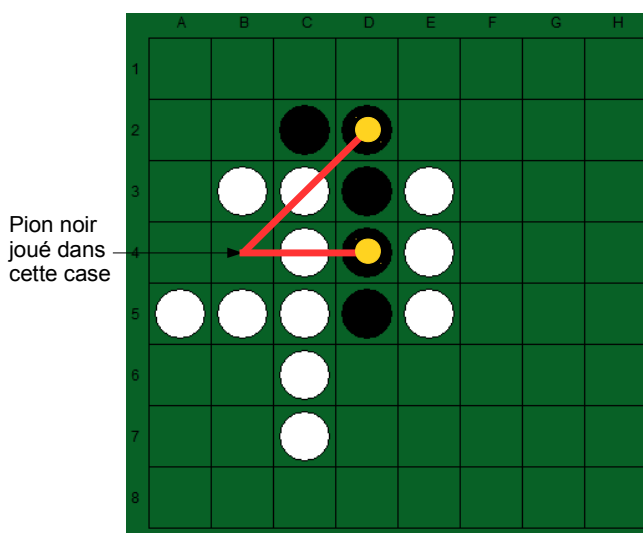
La fonction (image ci-dessous de droite) va ainsi effectuer le parcours montré sur le schéma de gauche après avoir défini les coordonnées maximales et minimales en « x » et en « y » de la zone de 3x3 entourant la position jouée.

4 - Dernier cas : le placement de pion ne retourne pas

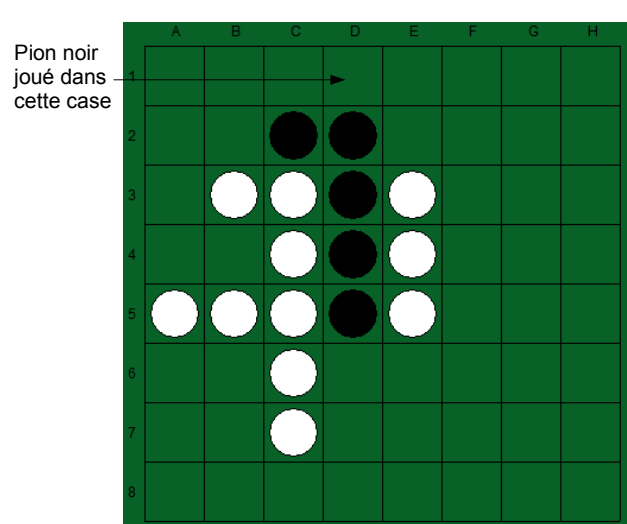
```
#Verification pour chaque case de ce carre de 3x3 entourant le pion
for temporaryY in range(yMin, yMax + 1):
    for temporaryX in range(xMin, xMax + 1):
        if('Le pion est un pion de l'adversaire'): return True
```

de pions adverses. Celui-ci est plus complexe, il est nécessaire de simuler le placement de pion afin de savoir s'il y a des pions retournés. La fonction utilisée pour effectuer cette opération (« [detectPawn\(color, x, y\)](#) ») va renvoyer une liste de position des « pions en bout de ligne » (en jaune sur le schéma ci-dessous) d'une ligne à retourner. De cette manière, une position ne retournant aucun pion sera détectée par une liste de positions vides.

```
except IndexError: return 2 #Si la liste est vide, aucun retournement n'est possible, on ne joue pas
```



Retournements possibles
Coordonnées renvoyées par la fonction



Retournements possibles
Coordonnées renvoyées par la fonction

C - Retournement des pions :

Connaissant les placements sans issue, nous pouvons en déduire ceux qui en ont. Dans notre cas cela revient à retourner les pions de l'adversaire « coincés » entre nos pions. Comme vu précédemment la fonction « [detectPawn\(color, x, y\)](#) » va nous donner la liste des « pions en bout de ligne » des lignes à retourner.

Il suffit donc de remplacer les lignes représentées en rouge sur le dernier schéma par des pions de la couleur jouée. Cette ligne est définie par deux points : l'endroit où le pion est joué et une position de la liste donnée par « [detectPawn](#) ». En calculant la différence des deux coordonnées sur l'axe « x » et sur l'axe « y », on peut déterminer la direction de cette ligne mais aussi se déplacer sur celle-ci. C'est exactement ce que fait la fonction « reverse » ([ligne 88](#)). Elle commence par se positionner aux coordonnées du pion joué puis va se déplacer selon la direction définie (donc sur la ligne) pour remplacer tous les pions qu'elle rencontre par la couleur jouée jusqu'à arriver aux coordonnées de fin de la ligne.

```
while(temporaryX != coordinateTo[0] or temporaryY != coordinateTo[1]): #Tant qu'on est pas arrive a destination
    grille[temporaryY][temporaryX] = color #On retourne le pion
    if(deltaX != 0): temporaryX = int(temporaryX + copysign(1, deltaX)) #On se decale d'un en X
    if(deltaY != 0): temporaryY = int(temporaryY + copysign(1, deltaY)) #On se decale d'un en Y
```

Nous savons, à ce stade, comment modifier la grille suite au placement d'un pion :

- Aucun changement effectué car le placement était incorrect.
- Retournement des pions adverses « emprisonnés » par le joueur.

Il est nécessaire de communiquer à l'interface (qui a appelé la fonction « place » pour positionner le pion) ce qu'il s'est produit pour qu'elle puisse agir en conséquence. Par exemple un placement invalide ne devra pas compter comme un pion placé pour permettre à l'utilisateur de jouer à nouveau sur une case valide. C'est pour cela que cette fonction renvoie un entier compris entre 0 et 2 :

- 0 : Le placement est correct, le pion est joué.
- 1 : La case est déjà occupée.
- 2 : Le pion est dans une position invalide (hors de la grille ou sans pions adverses dans le carré l'entourant).

VI) Détail des fonctions utilisées et fonctions d'informations :

A - GetColor ([ligne 38](#)) :

Cette fonction est utilisée afin de connaître la couleur d'un pion à un emplacement donné. Pour y parvenir, elle va renvoyer le nombre contenu par le tableau « grille » aux positions « x » et « y ».

B - GetNumberColor ([ligne 24](#)):

L'interface nous affiche le nombre de pions de chaque couleur, il a donc fallu implémenter une fonction permettant ce comptage : c'est « getNumberColor » qui opère en parcourant la grille à la recherche de la couleur demandée. Chaque fois que celle-ci est rencontrée, une valeur est incrémentée représentant ainsi le nombre de pions d'une certaine couleur.

C - DetectPawn (ligne 53) :

La fonction « detectPawn » est la fonction permettant de donner la liste des « pions en bout de ligne » d'une série à retourner. Son fonctionnement est assez simple. Celle-ci va créer une liste vide puis ajouter les réponses de la fonction « [getNextPawn\(color, directionX, directionY, x, y\)](#) » pour chaque direction possible (haut, bas, gauche, droite, ainsi que les diagonales) .

D - GetNextPawn (ligne 75) :

« getNextPawn » est sûrement la fonction la plus compliquée mais est essentielle au programme. C'est elle qui va générer les coordonnées des « pions en bout de ligne ». Ses arguments sont les suivants :

- color : La couleur du pion joué.
- directionX : Le déplacement selon « x » (soit négatif soit positif).
- directionY : Le déplacement selon « y » (soit négatif soit positif).
- x : La coordonnée « x » du point de départ (pion joué).
- y : La coordonnée « y » du point de départ (pion joué).

Par exemple, une directionX = 1 et directionY = 1, correspondra à la diagonale en bas à droite.

Connaissant ces informations et la grille du jeu, il va falloir déterminer si, dans la direction donnée, et en partant du pion origine, les pions rencontrés sont uniquement ceux de l'adversaire et que le dernier est de la couleur jouée. Bien sûr si un vide est rencontré durant le « trajet », cette direction n'est pas bonne.

La fonction va agir grâce à une boucle répétée au maximum huit fois (taille de la grille). Grâce aux directions, celle-ci peut générer chaque position se trouvant sur la ligne souhaitée (ci-dessous la partie de la fonction correspondante).

```
for temporaryPos in range(1, 8):
    tempX = int(x + copysign(abs(directionX) * temporaryPos, directionX))
    tempY = int(y + copysign(abs(directionY) * temporaryPos, directionY))
```

Prenons l'exemple de la composante « x » des coordonnées (cela fonctionne de la même manière pour « y ») . « [copysign](#) » est une fonction de python permettant de copier le signe d'un nombre sur un autre. Celle-ci prend deux paramètres, le premier nommé « a » et le deuxième « b ». On obtient donc au final « a » avec le signe de « b ». Dans notre cas cette fonction est utilisée pour appliquer le signe de « directionX » sur « le décalage relatif en « x » ». Le résultat est ainsi le nombre de cases dont il faut se déplacer selon « x » par rapport au point de départ. Puisque l'on ajoute le « x » de l'origine au résultat précédent, on obtient une nouvelle coordonnée « x » d'un point sur la ligne.

Maintenant que nous sommes capable de générer les coordonnées en partant de l'origine de chaque point composant la ligne désirée, il suffit d'effectuer nos tests pour savoir si celle-ci est valide.

Voici la liste des tests effectués :

- Si la case est hors de la grille, on stoppe la fonction. Cela veut dire que l'on a pas réussi à « emprisonner » un pion ennemi sans sortir du plateau.
- Si on tombe sur l'un de nos pions et que l'on a déjà croisé au moins un pion adverse, on retourne les coordonnées. Il s'agit d'un « pion bout de ligne ».

- Si l'on tombe sur l'un de nos pions mais que l'on a pas croisé de pions ennemis, on s'arrête ici car aucun pion n'est « emprisonné ».
- Si on tombe sur une case vide, on stoppe la fonction. La ligne ne respecte pas les règles.
- Si on tombe sur un pion ennemi, on continue la fonction en gardant en mémoire ce passage sur un pion ennemi.

VI - Conclusion :

Pour conclure sur la partie algorithme du programme, on peut constater que celle-ci réalise son travail. La processus de création de cette partie n'a pas été spécialement difficile mais tenait au début beaucoup de place (chaque vérification de lignes étaient séparées). Cependant un travail de réflexion a permis de rassembler ces différentes parties et en arriver à la fonction « getNextPawn » tel qu'elle est actuellement.

D'un point de vue plus général la création d'un tel projet nous a permis de réfléchir sur les méthodes de fonctionnement des matériels informatiques. Ces dernières se sont révélées plus complexes que l'on ne le pensait et nous ont permis ainsi d'acquérir une nouvelle façon de voir les choses. Pour mener à bien le projet, nous avons dû apprendre à travailler en équipe, pour pouvoir surmonter les obstacles en mettant en commun chacune de nos connaissances respectives.

VII – Annexes

A – Game.py

```
'''
@author: Thomas Couchoud
'''

from math import copysign

colorOne, colorTwo = 1, 2

def init():
    """
    Permet d'initialiser la grille du jeu
    """
    #Genere une grille vide
    for y in range(8):
        for x in range(8):
            grille[y][x] = 0

    #Initialisation de la grille de depart
    grille[3][3] = colorOne
    grille[4][3] = colorTwo
    grille[4][4] = colorOne
    grille[3][4] = colorTwo

def getNumberColor(color):
    """
    Permet de compter le nombre de pions sur la grille (Couleur 0 pour savoir le nombre de
```



```

cases vides)
Arguments:
    color -> La couleur du pion a compter
Return:
    Le nombre de pions de la couleur demandee presents sur la grille
"""

count = 0
for y in range(0, 8):
    for x in range(0, 8):
        if(getColor(x, y) == color): count += 1
return count

def getColor(x, y):
    """
    Permet de connaitre le pion sur la case
    Arguments:
        x -> La position x du pion (horizontale)
        y -> La position y du pion (verticale)
    Return:
        0 -> Case vide
        1 -> Couleur 1
        2 -> Couleur 2
        3 -> Hors de la grille
    """
    if(x < 0 or x > 7 or y < 0 or y > 7): return 3 #Hors de la grille
    return grille[y][x]

def detectPawn(color, x, y):
    """
    Permet de connaitre les pions de meme couleur qui sont sur les memes lignes
    Arguments:
        color -> La couleur du pion
        x -> La position x du pion (horizontale)
        y -> La position y du pion (verticale)
    Return:
        Une liste de tableaux bi-dimensionnels contenant les positions des pions trouves
    """
    pawnList = []
    pawnList.append(getNextPawn(color, 1, 0, x, y)) #Horizontal +
    pawnList.append(getNextPawn(color, -1, 0, x, y)) #Horizontal -
    pawnList.append(getNextPawn(color, 0, 1, x, y)) #Vertical +
    pawnList.append(getNextPawn(color, 0, -1, x, y)) #Vertical -
    pawnList.append(getNextPawn(color, 1, 1, x, y)) #Diagonale ++
    pawnList.append(getNextPawn(color, 1, -1, x, y)) #Diagonale +-
    pawnList.append(getNextPawn(color, -1, 1, x, y)) #Diagonale -+
    pawnList.append(getNextPawn(color, -1, -1, x, y)) #Diagonale --
    print(pawnList)
    return pawnList

def getNextPawn(color, directionX, directionY, x, y):
    opponentPawnMet = False
    if(directionX == 0 and directionY == 0): return #Aucun decalage effectue, on arrete
    for temporaryPos in range(1, 8):
        tempX = int(x + copysign(abs(directionX) * temporaryPos, directionX))
        tempY = int(y + copysign(abs(directionY) * temporaryPos, directionY))
        if(getColor(tempX, tempY) == 3): return #Hors de la grille
        elif(getColor(tempX, tempY) == color and (tempX != x or tempY != y) and
opponentPawnMet): #On est passe par des pions adverse et rencontrons un de nos pions,
ligne finie
            return (tempX, tempY)
        elif(getColor(tempX, tempY) == color and (tempX != x or tempY != y) and not

```

opponentPawnMet): **return** #C'est un de nos pions mais on a pas rencontre de pions adverse, pas de ligne

elif(getColor(tempX, tempY) == 0 **and** (tempX != x **or** tempY != y)): **return** #On est tombe sur une case vide, pas de ligne

elif(getColor(tempX, tempY) != color **and** getColor(tempX, tempY) != 0 **and** getColor(tempX, tempY) != 3): opponentPawnMet = **True** #Ce n'est pas notre pion, on continue la ligne

def reverse(color, coordinateBase, coordinateTo):

"""

Permet de savoir si le point 2 est sur une diagonale du point 1

Arguments:

color -> La couleur a mettre

coordinateBase -> Les coordonnees du pion origine

coordinateTo -> Les coordonnees du point d'arrive

"""

deltaX = coordinateTo[0] - coordinateBase[0] #Decalage en X

deltaY = coordinateTo[1] - coordinateBase[1] #Decalage en Y

temporaryX = coordinateBase[0] #X de depart

temporaryY = coordinateBase[1] #Y de depart

while(temporaryX != coordinateTo[0] **or** temporaryY != coordinateTo[1]): #Tant qu'on est pas arrive a destination

grille[temporaryY][temporaryX] = color #On retourne le pion

if(deltaX != 0): temporaryX = **int**(temporaryX + copysign(1, deltaX)) #On se decale d'un en

X

Y

if(deltaY != 0): temporaryY = **int**(temporaryY + copysign(1, deltaY)) #On se decale d'un en

Y

grille[coordinateTo[1]][coordinateTo[0]] = color #On retourne le pion

def hasPawnNext(color, x, y):

"""

Permet de savoir si un pion est entoure d'autres pions d'une autre couleur

Arguments:

x -> La position x du pion (horizontale)

y -> La position y du pion (verticale)

Return:

True -> Le pion est entoure d'autres pions

False -> Le pion n'est pas entoure d'autres pions

"""

xMin = x - 1 #XMin du carre de 3x3 entourant le pion

xMax = x + 1 #XMax du carre de 3x3 entourant le pion

if(xMin < 0): xMin = 0 #Si le XMin sort de la grille, on le remet dedans

if(xMax > 7): xMax = 7 #Si le XMax sort de la grille, on le remet dedans

yMin = y - 1 #YMin du carre de 3x3 entourant le pion

yMax = y + 1 #YMax du carre de 3x3 entourant le pion

if(yMin < 0): yMin = 0 #Si le YMin sort de la grille, on le remet dedans

if(yMax > 7): yMax = 7 #Si le YMax sort de la grille, on le remet dedans

#Verification pour chaque case de ce carre de 3x3 entourant le pion

for temporaryY **in** range(yMin, yMax + 1):

for temporaryX **in** range(xMin, xMax + 1):

if(getColor(temporaryX, temporaryY) != 0 **and** getColor(temporaryX, temporaryY) != color **and** (temporaryX != x **or** temporaryY != y)): **return** **True** #Si la case n'est pas vide & Si la case n'est pas notre couleur & Si ce n'est pas la ou on joue le pion, c'est bon, il y a pion adverse autour

return **False**

def place(color, x, y):

"""

Permet de placer un pion

Arguments:

```

    color -> La couleur du pion joue (1 = blanc, 2 = noir)
    x -> La position x du pion (horizontale)
    y -> La position y du pion (verticale)
Return:
    0 -> Pas d'erreur detectee
    1 -> Case deja occupee
    2 -> Impossible de placer le pion
"""

if(getColor(x, y) == 3 or not hasPawnNext(color, x, y)): return 2 #Si en dehors de la grille ou
n'a pas de pion adverse a cote, on ne peut pas jouer
elif(getColor(x, y) != 0): return 1 #Si la case n'est pas vide, on ne peut pas jouer
try:
    pawnList = detectPawn(color, x, y) #On recupere la liste des lignes a retourner
    while(pawnList.count(None) > 0): pawnList.remove(None) #Retire les 'None' de la liste
    if(len(pawnList) < 1): return 2 #Si la liste est vide, aucun retournement n'est possible, on
ne joue pas
    for l in pawnList: #Pour chque position de fin de ligne
        reverse(color, (x, y), l) #On retourne cette ligne avec la couleur du joueur
    except IndexError: return 2 #Si la liste est vide, aucun retournement n'est possible, on ne
joue pas
    grille[y][x] = color #On place le pion joue
    return 0 #On notifie que le jeu est bon

def showGrid():
    """
    Permet d'afficher la grille dans la console
    """
    for l in grille:
        print(l)
    print()

#Lance a l'import des fonctions
grille = [[0 for x in range(8)] for x in range(8)] #Creation de l'objet grille
init() #On initialise la grille

```

B - Interface.py :

```

# -*- coding: utf-8 -*-
"""
@author: Olivier Froger & Johann Jacques
"""

import tkinter.messagebox
import time
from tkinter import Tk, Toplevel, Canvas, Button, DISABLED, END, Entry, Label, Menu,
NORMAL, PhotoImage, Radiobutton, StringVar, Text
from Game import init, getColor, place, getNumberColor
from time import localtime, strftime
import socket
import select
import threading

#Johann
global serverThread, clientThread, fen
server = False
serverThread = None
IP = "192.168.0.12"

#Johann
class ThreadClient(threading.Thread):
    def __init__(self, tHote, tPort):

```

```

#Initialisation des variables
threading.Thread.__init__(self)
self.hote = tHote
self.port = tPort
self.nom = "TClient"
self.aEnvoyer = ""
self.Terminated = False
self.start()

def run(self):
    self.timer = time.time()
    connexionServeur = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Creation
de la connexion au serveur
    connexionServeur.connect((self.hote, self.port)) #Connexion au serveur
    print("\nConnexion etablie avec le serveur sur le port " + str(self.port))
    message = ""
    while not self.Terminated: #Tant qu'on ne se deconnecte pas
        try:
            if message != self.aEnvoyer: #Si il y a un nouveau message a envoyer
                message = self.aEnvoyer #On recupere le message
                connexionServeur.send(message.encode()) #On envoie le message
                messageRecu = connexionServeur.recv(1024).decode() #On recupere la reponse
du serveur
                print("Reponse serveur: " + str(messageRecu))
            if time.time() - self.timer > 0.5: #Si il faut actualiser les donnees
                self.timer = time.time() #On remet a zero le temps
                connexionServeur.send("£9".encode()) #On envoie une requete 9
                messageRecu = connexionServeur.recv(1024).decode() #On recupere la reponse
du serveur
                print("Reponse 9 serveur: " + str(messageRecu))
                fen.decrypt(messageRecu) #On decrypte la reponse
            except "ConnectionAbortedError":
                print()
            print("Fermeture de la connexion")
            connexionServeur.close() #Fermeture de la connexion

    def stop(self):
        self.Terminated = True #Arret du client (sors de la boucle while)

    def setAEnvoyer(self, message):
        self.aEnvoyer = message #Modifie le message a envoyer

#Johann
def envoi(messageType, *args): #Cree un message de la forme [0-9][*]
    global clientThread
    message = str('£' + messageType)
    for arg in args:
        message += '&' + str(arg)
    clientThread.setAEnvoyer(message)

#Johann
def arret_client():
    global clientThread
    clientThread.stop() #Arrete le client

#Johann
def lancement_client(hote = IP, port = 50000):
    global clientThread
    clientThread = ThreadClient(hote, port)

#Olivier

```

```

class Interface:
    def a_accent_maj(self):
        """
        Permet d'afficher un a accent majuscule
        """
        return chr(0x00c0)

    def a_accent(self):
        """
        Permet d'afficher un a accent
        """
        return chr(0x00e0)

    def e_aigu(self):
        """
        Permet d'afficher un e accent aigu
        """
        return chr(0x00e9)

    def e_grave(self):
        """
        Permet d'afficher un e accent grave
        """
        return chr(0x00e8)

    def e_circonflexe(self):
        """
        Permet d'afficher un e accent circonflexe
        """
        return chr(0x00ea)

    #Olivier
    def initialisation(self):
        """
        Permet d'initialiser par default
        """
        init()
        self.canvasGrille.delete("pion")
        self.lastPlayed = self.blanc
        self.refresh()

    #Olivier
    def refreshBG(self):
        """
        Permet de mettre a jour le fond de jeu
        """
        if self.backgroundPrefs.get() == "none":
            self.canvasGrille.itemconfigure(self.background, image = None)
        elif self.backgroundPrefs.get() == "bear":
            self.canvasGrille.itemconfigure(self.background, image = self.photoBackgroundBear)
        elif self.backgroundPrefs.get() == "wood":
            self.canvasGrille.itemconfigure(self.background, image = self.photoBackgroundWood)
        elif self.backgroundPrefs.get() == "digital":
            self.canvasGrille.itemconfigure(self.background, image = self.photoBackgroundDigital)
        elif self.backgroundPrefs.get() == "abstract":
            self.canvasGrille.itemconfigure(self.background, image = self.photoBackgroundAbstract)
        elif self.backgroundPrefs.get() == "wall":
            self.canvasGrille.itemconfigure(self.background, image = self.photoBackgroundWall)
        elif self.backgroundPrefs.get() == "penguins":
            self.canvasGrille.itemconfigure(self.background, image =
self.photoBackgroundPenguins)

```

```

elif self.backgroundPrefs.get() == "koala":
    self.canvasGrille.itemconfigure(self.background, image = self.photoBackgroundKoala)
elif self.backgroundPrefs.get() == "desert":
    self.canvasGrille.itemconfigure(self.background, image = self.photoBackgroundDesert)

#Olivier
def refresh(self):
    """
    Permet de rafraichir les informations de jeu
    """
    for x in range(0, 8):
        for y in range(0, 8):
            self.placer_pion(getColor(x, y), x, y)
    self.count1Var.set(getNumberColor(self.blanc))
    self.count2Var.set(getNumberColor(self.noir))

#Olivier
def placer_pion(self, color, x, y):
    """
    Permet de placer un pion

    Arguments:
        self.color -> Le numero de la couleur jouee
        x -> La position x (Horizontale)
        y -> La position y (verticale)
    """
    offsetGrid, borderLineColor, backgroundColor = 5, "black", self.colorVert
    if color == self.blanc: backgroundColor = self.colorPion1Prefs.get()
    elif color == self.noir: backgroundColor = self.colorPion2Prefs.get()
    else: return
    self.canvasGrille.create_oval(self.gridOffsetCanvas + (x * self.tailleCase) + offsetGrid,
self.gridOffsetCanvas + (y * self.tailleCase) + offsetGrid, self.gridOffsetCanvas + (self.tailleCase
* (x + 1)) - offsetGrid, self.gridOffsetCanvas + (self.tailleCase * (y + 1)) - offsetGrid, tags =
"pion", fill = backgroundColor, outline = borderLineColor)

#Olivier
def mettre_pion(self, color, x, y):
    if(color != self.lastPlayed):
        if(place(color, x, y) == 0):
            if(not server): envoi("0", color, x, y)
            self.caseX = x
            self.caseY = y
            self.lastPlayed = color
            self.refresh()

#Olivier
def clic_pion(self, event):
    """
    Recupere un clic souris et joue le pion dans la case appropriee

    Arguments:
        event -> L'event du clic
    """
    self.mettre_pion(self.colorPlayer, ((event.x) - self.gridOffsetCanvas) // self.tailleCase,
((event.y) - self.gridOffsetCanvas) // self.tailleCase)

#Olivier
def regles(self):
    """
    Permet d'afficher la fenetrePrincipale des regles
    """

```

```

self.fenetreRegles = Tk()
self.fenetreRegles.title("R" + self.e_grave() + "gles du jeu")
self.fenetreRegles.geometry("500x500")
self.fenetreRegles.resizable(0, 0)
self.fenetreRegles.mainloop()

#Olivier
def preferences(self):
    """
    Permet d'afficher la fenetrePrincipale des preferences
    """
    #global self.fenetrePreferences, self.colorsListP1, self.colorsListP2
    self.fenetrePreferences = Toplevel()
    self.fenetrePreferences.grab_set()
    self.fenetrePreferences.title("Pr" + self.e_aigu() + "f" + self.e_aigu() + "rences")
    self.fenetrePreferences.geometry("500x500")
    self.fenetrePreferences.resizable(0, 0)
    self.canvasPion1Prefs = Canvas(self.fenetrePreferences, bg = "gray", height = 250, width
= 250)
    self.canvasPion2Prefs = Canvas(self.fenetrePreferences, bg = "gray", height = 250, width
= 250)
    self.canvasPlateauPrefs = Canvas(self.fenetrePreferences, bg = "gray", height = 250,
width = 500)
    self.canvasPion1Prefs.place(x = 250, y = 0)
    self.canvasPion2Prefs.place(x = 0, y = 0)
    self.canvasPlateauPrefs.place(x = 0, y = 250)
    Label(self.canvasPion1Prefs, bg = "gray", font = self.Comic2, text = "Pion 2").place(x = 10,
y = 10)
    Label(self.canvasPion2Prefs, bg = "gray", font = self.Comic2, text = "Pion 1").place(x = 10,
y = 10)
    Label(self.canvasPlateauPrefs, bg = "gray", font = self.Comic2, text = "Plateau").place(x =
5, y = 5)
    self.colorsListP1 = [("blanc", "white"), ("noir", "black"), ("Orange", "orange"), ("Bleu", "blue"),
("Vert", "green"), ("Rouge", "red"), ("Jaune", "Yellow")]
    self.colorsListP2 = [("blanc", "white"), ("noir", "black"), ("Orange", "orange"), ("Bleu", "blue"),
("Vert", "green"), ("Rouge", "red"), ("Jaune", "Yellow")]
    self.backgroundsList = [("Aucun", "none"), ("Bois", "wood"), ("Nounours", "bear"), ("Digital",
"digital"), ("Abstrait", "abstract"), ("Mur", "wall"), ("Penguins", "penguins"), ("Koala", "koala"),
("Desert", "desert")] # TODO add backgrounds ("nomAAfficher", "ID")
    tempValue = 0
    for tempText, tempColor in self.colorsListP1:
        Radiobutton(self.canvasPion1Prefs, text = tempText, bg = "gray", value = tempColor,
variable = self.colorPion1Prefs).place(x = 10, y = 60 + 20 * tempValue)
        tempValue += 1
    tempValue = 0
    for tempText, tempColor in self.colorsListP2:
        Radiobutton(self.canvasPion2Prefs, text = tempText, bg = "gray", value = tempColor,
variable = self.colorPion2Prefs).place(x = 10, y = 60 + 20 * tempValue)
        tempValue += 1
    tempValue = 0
    for tempText, tempColor in self.backgroundsList:
        Radiobutton(self.canvasPlateauPrefs, text = tempText, bg = "gray", value = tempColor,
variable = self.backgroundPrefs).place(x = 10, y = 60 + 20 * tempValue)
        tempValue += 1
    Button(self.canvasPlateauPrefs, text = "Valider", command =
self.appli_preferences).place(x = 230, y = 200)
    self.fenetrePreferences.mainloop()

#Olivier
def appli_preferences(self):
    """

```



```

    Permet d'appliquer les changements de preferences a l'interface
    """
    #global self.fenetrePreferences, self.colorPion1Prefs, self.colorPion2Prefs
    if self.colorPion1Prefs.get() == self.colorPion2Prefs.get():
        tkinter.messagebox.showerror("Erreur", "Vous ne pouvez pas utiliser la meme couleur
pour les deux pions")
        return
    else:
        self.count1Label.config(fg = self.colorPion1Prefs.get())
        self.count2Label.config(fg = self.colorPion2Prefs.get())
        self.count1Label.config(textvariable = self.count1Var)
        self.count2Label.config(textvariable = self.count2Var)
        self.refresh()
        self.refreshBG()
        self.fenetrePreferences.destroy()

#Olivier
def a_propos(self):
    """
    Permet d'afficher la fenetrePrincipale a propos
    """

    self.fenetreAPropos = Tk()
    self.fenetreAPropos.title(self.a_accent_maj() + " propos de")
    self.fenetreAPropos.geometry("500x500")
    self.fenetreAPropos.resizable(0, 0)
    self.fenetreAPropos.mainloop()

#Olivier
def parler(self, event = None):
    """
    Permet au joueur d'envoyer un message dans le chat
    """

    #print(self.decrypt(self.chatEntry.get()))
    self.textTraitement(self.chatEntry.get(), "player", str(self.pseudoEntry.get()), self.chatColor)
    self.chatEntry.delete(0, END)

#Olivier
def isValidPseudo(self, pseudo):
    if(pseudo == ""): return False
    character = ['|', '°', '§', '£', 'µ', '&', '£']
    for z in character:
        if(str(pseudo).find(z) > -1):
            return False
    return True

#Olivier
def textTraitement(self, text, user, name, color):
    """
    Permet d'afficher du texte dans la zone appropriée

    Arguments:
        text -> Le texte a afficher
        user -> La parsonne qui parle (Joueur: "player", Systeme: "system", Adversaire:
"opponent", Spectateur: "spec")
        name -> Le nom a afficher
        self.color -> La couleur du texte
    """

    self.text=str(self.chatEntry.get())
    self.textTime = strftime('%H:%M:%S : ', localtime())
    self.name = str(user)
    self.text = self.textTime + str(self.name) + " -> " + str(self.text)

```



```

message2 = str(text)
validPseudo = self.isValidPseudo(self.pseudoEntry.get())
if user == "player" and not validPseudo:
    return
else:
    if(str(text) == "" or len(text) == text.count(" ")): return
    message2 = str(text)
    textTime = strftime('%H:%M:%S : ', localtime())
    text = textTime + str(name) + " -> " + str(text)
    self.textChat.config(state = NORMAL)
    self.textChat.insert(0.0, text + "\n")
    self.textChat.tag_configure(color, foreground = color)
    self.textChat.tag_add(color, "1." + str(len(textTime + str(name) + " -> ")), "1." +
str(len(text)))
    colorUser = "black"
    if(user == "player"): colorUser = "black"
    elif(user == "system"): colorUser = "red"
    else: colorUser = "black"
    self.textChat.tag_configure("name", foreground = colorUser)
    self.textChat.tag_add("name", "1." + str(len(textTime)), "1." + str(len(textTime + name)))
    self.joueur = str(user)
    if server and user == "player":
        self.message.append(message2)
        print(self.message)
    elif user == "player":
        envoi("1", name, message2)
    self.textChat.config(state = DISABLED)

#Olivier
def connexion(self):
    """
    Permet d'initialiser la connexion
    """
    Label(self.canvasInfos, text = " " + str(self.pseudoEntry.get()), bg = self.colorVert, fg =
'red').place(x = 50, y = 2)
    pseudo = self.pseudoEntry.get()
    if pseudo == "" or self.pseudoEntry.get().find('&') > -1 or self.pseudoEntry.get().find('£') >
-1:
        tkinter.messagebox.showerror("Erreur", "Vous devez rentrer un psuedo valide")
    else:
        self.pseudoEntry.place_forget()
        self.connexionButton.place_forget()
        self.textChat.config(state = NORMAL)
        self.textChat.config(fg = "red")
        self.textTraitment("Vous " + self.e_circonflexe() + "tes connect" + self.e_aigu() + " en tant
que " + str(self.pseudoEntry.get()), "system", "Syst" + self.e_grave() + "me", "red")

#Olivier
def stopInterface(self):
    if tkinter.messagebox.askquestion("", "Voulez vous vraiment quitter ?")==="no": return
    else: self.fenetrePrincipale.destroy()
    #arret_serv()

#Olivier
def getLastPos(self):
    print(self.lastPlayed, self.caseX, self.caseY)
    return [self.lastPlayed, self.caseX, self.caseY]

#Olivier
def getLastChat(self):
    if(len(self.message) > 0):

```

```

        m = self.message
        m.append(self.pseudoEntry.get())
        self.message = []
        return m
    return None

def passer(self):
    if(self.lastPlayed != self.colorPlayer):
        self.lastPlayed = self.colorPlayer
        if(server): self.requet = "£7"
        else: envoi("7")

#Johann
def decrypt(self, x):
    try:
        if x[0:2] == '£0': #£0&col&x&y
            self.mettre_pion(int(x[3]), int(x[5]), int(x[7]))
            return "OK " + str(x[3]) + str(x[5]) + str(x[7])
        elif x[0:2] == '£1': #£1&user&mess
            x = x[x.find('&') + 1:]
            user = x[:x.find('&')]
            message = x[x.find('&') + 1:]
            self.textTraitment(message, "opponent", user, 'red')
            return "OK " + message + "opponent" + user + 'red'
        elif x[0:2] == "£7":
            self.textTraitment("A votre tour de jouer", "system", "Syst" + self.e_grave() + "me",
"red");
            self.lastPlayed = 0
            return "OK 7"
        elif x[0:2] == '£8':
            x = x[2:]
            while x.find('£') > -1:
                x = x[x.find('£') + 1:]
                if(x.find('£') > -1):
                    print(self.decrypt('£' + x[:x.find('£')]))
                    x = x[x.find('£'):]
                else:
                    print(fen.decrypt('£' + x))
            return "OK 8"
        elif x[0:2] == '£9':
            mess = '£8'
            s = self.getLastPos()
            b = True
            for a in s:
                b = b and (str(a) != "None")
            if b:
                mess += "£0&" + str(s[0]) + "&" + str(s[1]) + "&" + str(s[2])
            m = self.getLastChat()
            print(m)
            if(self.requet != None):
                mess += self.requet
                self.requet = None
            if(m != None):
                user = m.pop()
                for message in m:
                    mess += "£1&" + user + "&" + message
            print("OK 9")
            return mess
    except IndexError as e:
        print(e)
        pass

```

```

return "Error " + x

#Olivier
def __init__(self):
    #Initialisation des variables
    self.requet, self.caseX, self.caseY, self.colorPlayer, self.lastPlayed, self.colorVert,
self.blanc, self.noir, self.yOffsetCanvas, self.xOffsetCanvas, self.gridOffsetCanvas,
self.tailleCase, self.Comic, self.Comic2, self.Comic3, self.tourDeJeu, self.chatColor,
self.colorPlayerChat, self.colorPion1Prefs, self.colorPion2Prefs, self.color, self.pseudoEntry,
self.pseudo, self.fenetrePreferences, self.colorsListP1, self.colorsListP2, self.message= None,
9, 9, 0, 0, "#086126", 1, 2, 2, 8, 25, 50, ("self.Comic sans MS", "9"), ("self.Comic sans MS",
"25"), ("self.Comic sans MS", "35"), 1, "black",
"blue", None, None, None, None, None, None, None, None, None, []
    if(server): self.colorPlayer = self.noir
    else: self.colorPlayer = self.blanc
    self.lastPlayed = self.blanc

    #from Server import lancement_serv, arret_serv
    self.fenetreConnexion = Tk()
    self.fenetreConnexion.title("Connexion . . .")
    self.fenetreConnexion.geometry("350x100")
    #Label(self.fenetreConnexion, text = "Hote : ").place(x = 1, y = 1)
    #Label(self.fenetreConnexion, text = "Port : ").place(x = 1, y = 51)
    #Entry(self.fenetreConnexion).place(x = 50, y = 1)
    #Entry(self.fenetreConnexion).place(x = 50, y = 51)
    #Button(self.fenetreConnexion, text = "Connexion", command =
self.fenetreConnexion.destroy).place(x = 185, y = 30)
    self.fenetreConnexion.mainloop()
    #Initialisation de la fenetre
    self.fenetrePrincipale = Tk()
    self.fenetrePrincipale.title("Othello")
    self.fenetrePrincipale.geometry("800x450")
    self.fenetrePrincipale.resizable(0, 0)
    self.fenetrePrincipale.protocol("WM_DELETE_WINDOW", self.stopInterface)
    Label(self.fenetrePrincipale, text = "Othello", font = self.Comic2, bg =
self.colorVert).place(x = 570, y = 35)

    #Declaration des images de fond de jeu encodées en base 64 (coupée car les encodages
tiennent sur plus de 200 pages)

    #Creation et initialisation des variables de preferences
    self.colorPion1Prefs, self.colorPion2Prefs, self.backgroundPrefs = StringVar(), StringVar(),
StringVar()
    self.colorPion1Prefs.set("white")
    self.colorPion2Prefs.set("black")
    self.backgroundPrefs.set("none")

    #Creation de la barre de menus
    self.menubar = Menu(self.fenetrePrincipale)

    self.menufichier = Menu(self.menubar, tearoff = 0)
    self.menufichier.add_command(label = "Nouvelle partie", command = self.initialisation)
    self.menufichier.add_command(label = "Pr" + self.e_aigu() + "f" + self.e_aigu() + "rences",
command = self.preferences)
    self.menufichier.add_command(label = "Quitter", command =
self.fenetrePrincipale.destroy)

    self.menuaide = Menu(self.menubar, tearoff = 0)

```

```

        self.menuaide.add_command(label = "R" + self.e_grave() + "gles du jeu", command =
self.regles)
        self.menuaide.add_command(label = self.a_accent_maj() + " propos de", command =
self.a_propos)

        self.menubar.add_cascade(label = "Fichier", menu = self.menufichier)
        self.menubar.add_cascade(label = "Aide", menu = self.menuaide)

        self.fenetrePrincipale.config(menu = self.menubar)

        #Creation du canvas contenant la grille de jeu
        self.canvasGrille = Canvas(self.fenetrePrincipale, bg = self.colorVert, height = 450, width =
435)
        self.canvasGrille.place(x = 0, y = 0)
        self.canvasGrille.bind("<Button-1>", self.clic_pion)
        Label(self.canvasGrille, text = "A", font = self.Comic, bg = self.colorVert).place(x = 45, y =
self.yOffsetCanvas)
        Label(self.canvasGrille, text = "B", font = self.Comic, bg = self.colorVert).place(x = 95, y =
self.yOffsetCanvas)
        Label(self.canvasGrille, text = "C", font = self.Comic, bg = self.colorVert).place(x = 145, y =
self.yOffsetCanvas)
        Label(self.canvasGrille, text = "D", font = self.Comic, bg = self.colorVert).place(x = 195, y =
self.yOffsetCanvas)
        Label(self.canvasGrille, text = "E", font = self.Comic, bg = self.colorVert).place(x = 245, y =
self.yOffsetCanvas)
        Label(self.canvasGrille, text = "F", font = self.Comic, bg = self.colorVert).place(x = 295, y =
self.yOffsetCanvas)
        Label(self.canvasGrille, text = "G", font = self.Comic, bg = self.colorVert).place(x = 345, y =
self.yOffsetCanvas)
        Label(self.canvasGrille, text = "H", font = self.Comic, bg = self.colorVert).place(x = 395, y =
self.yOffsetCanvas)
        Label(self.canvasGrille, text = "1", font = self.Comic, bg = self.colorVert).place(x =
self.xOffsetCanvas, y = 40)
        Label(self.canvasGrille, text = "2", font = self.Comic, bg = self.colorVert).place(x =
self.xOffsetCanvas, y = 90)
        Label(self.canvasGrille, text = "3", font = self.Comic, bg = self.colorVert).place(x =
self.xOffsetCanvas, y = 140)
        Label(self.canvasGrille, text = "4", font = self.Comic, bg = self.colorVert).place(x =
self.xOffsetCanvas, y = 190)
        Label(self.canvasGrille, text = "5", font = self.Comic, bg = self.colorVert).place(x =
self.xOffsetCanvas, y = 240)
        Label(self.canvasGrille, text = "6", font = self.Comic, bg = self.colorVert).place(x =
self.xOffsetCanvas, y = 290)
        Label(self.canvasGrille, text = "7", font = self.Comic, bg = self.colorVert).place(x =
self.xOffsetCanvas, y = 340)
        Label(self.canvasGrille, text = "8", font = self.Comic, bg = self.colorVert).place(x =
self.xOffsetCanvas, y = 390)

        self.background = self.canvasGrille.create_image(226, 226)
        self.refreshBG()

        x1, x2, y1, y2 = self.gridOffsetCanvas, self.gridOffsetCanvas, self.gridOffsetCanvas,
self.gridOffsetCanvas
        for i in range(0, 9):
            self.canvasGrille.create_line(x1, y1, x1 + 400, y1)
            self.canvasGrille.create_line(x2, y2, x2, y2 + 400)
            y1 += self.tailleCase
            x2 += self.tailleCase

```

```

#Creation du canvas contenant les infos de jeu et le chat
self.canvasInfos = Canvas(self.fenetrePrincipale, bg = self.colorVert, height = 450, width =
365)
self.canvasInfos.place(x = 435, y = 0)
Label(self.canvasInfos, text = "Pseudo:", font = self.Comic, bg = self.colorVert).place(x = 5,
y = 2)
self.pseudoEntry = Entry(self.canvasInfos)
self.pseudoEntry.place(x = 50, y = 2)
self.connexionButton = Button(self.canvasInfos, text = "Connexion", command =
self.connexion)
self.connexionButton.place(x = 185, y = 2)
self.count1Var, self.count2Var = StringVar(), StringVar()
self.count1Label = Label(self.canvasInfos, textvariable = self.count1Var, font =
self.Comic3, fg = self.colorPion1Prefs.get(), bg = self.colorVert)
self.count2Label = Label(self.canvasInfos, textvariable = self.count2Var, font =
self.Comic3, fg = self.colorPion2Prefs.get(), bg = self.colorVert)
self.count1Label.place(x = 240, y = 70)
self.count2Label.place(x = 115, y = 70)
Button(self.canvasInfos, text = "Envoyer", command = self.parler).place(x = 270, y = 405)
self.chatEntry = Entry(self.canvasInfos, width = 35)
self.chatEntry.bind("<Return>", self.parler)
self.chatEntry.place(x = 40, y = 400)
self.textChat = Text(self.canvasInfos, state = DISABLED, width = 41, height = 8, font =
("self.Comic sans ms", 10))
self.textChat.config(fg = "black")
self.textChat.place(x = 20, y = 185)
Button(self.canvasInfos, text = "Nouvelle partie", command = self.initialisation).place(x =
270, y = 1)
Button(self.canvasInfos, text = "Passer son tour", command = self.passer).place(x = 270, y
= 30)
self.canvasInfos.create_line(100, 150, 180, 150)
self.canvasInfos.create_line(100, 65, 100, 150)
self.canvasInfos.create_line(100, 65, 180, 65)
self.canvasInfos.create_line(180, 65, 180, 150)
self.initialisation()
fen = Interface()

```

#Johann

```

class ThreadServer(threading.Thread): #Initialisation varrables
    def __init__(self, tHote = socket.gethostbyname(socket.gethostname()), tPort = 50000):
        threading.Thread.__init__(self)
        self.hote = tHote
        self.port = tPort
        self.nom = "TServer"
        self.Terminated = False
        self.start()

    def run(self):
        connexionPrincipale = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Creation
variable de connexion
        connexionPrincipale.bind((self.hote, self.port)) #On definit IP + port
        connexionPrincipale.listen(5) #On lance la l'ecoute de requetes
        server = "Bienvenue sur le serveur OTHELLO ({:}:{})".format(self.hote, self.port)
        print("\n" + server.upper().center(85) + "\nTraitement des donnees :\n")
        clientsConnectes = []
        while not self.Terminated:
            try:
                connexionsEntrantes, wlist, xlist = select.select([connexionPrincipale], [], [], 0.05) #On
rcupere les clients connectes sur le serveur
                for connexion in connexionsEntrantes: #Pour chacun d'entre eux, on accepte leur
connexion

```

```

        connexionClient, infosConnexion = connexion.accept()
        clientsConnectes.append(connexionClient)
        clientsALire = []
        clientsALire, wlist, xlist = select.select(clientsConnectes, [], [], 0.05) #On recupere la
liste des requetes
        for client in clientsALire: #Pour chaque requete
            messageRecu = client.recv(1024) #On recupere le message
            messageRecu = messageRecu.decode()
            print("> " + messageRecu)
            client.send(fen.decrypt(messageRecu).encode()) #On envoie la reponse au client
        except select.error:
            pass
        print("Fermeture des connexions")
        for client in clientsConnectes:
            client.close()
        connexionPrincipale.close()

    def stop(self):
        self.Terminated = True
#Johann
    def lancement_serv():
        global serverThread
        serverThread = ThreadServer()

#Johann
    def arret_serv():
        global serverThread
        if(serverThread != None):serverThread.stop()

    if(server):lancement_serv()
    else:lancement_client()
    fen.fenetrePrincipale.mainloop()

```