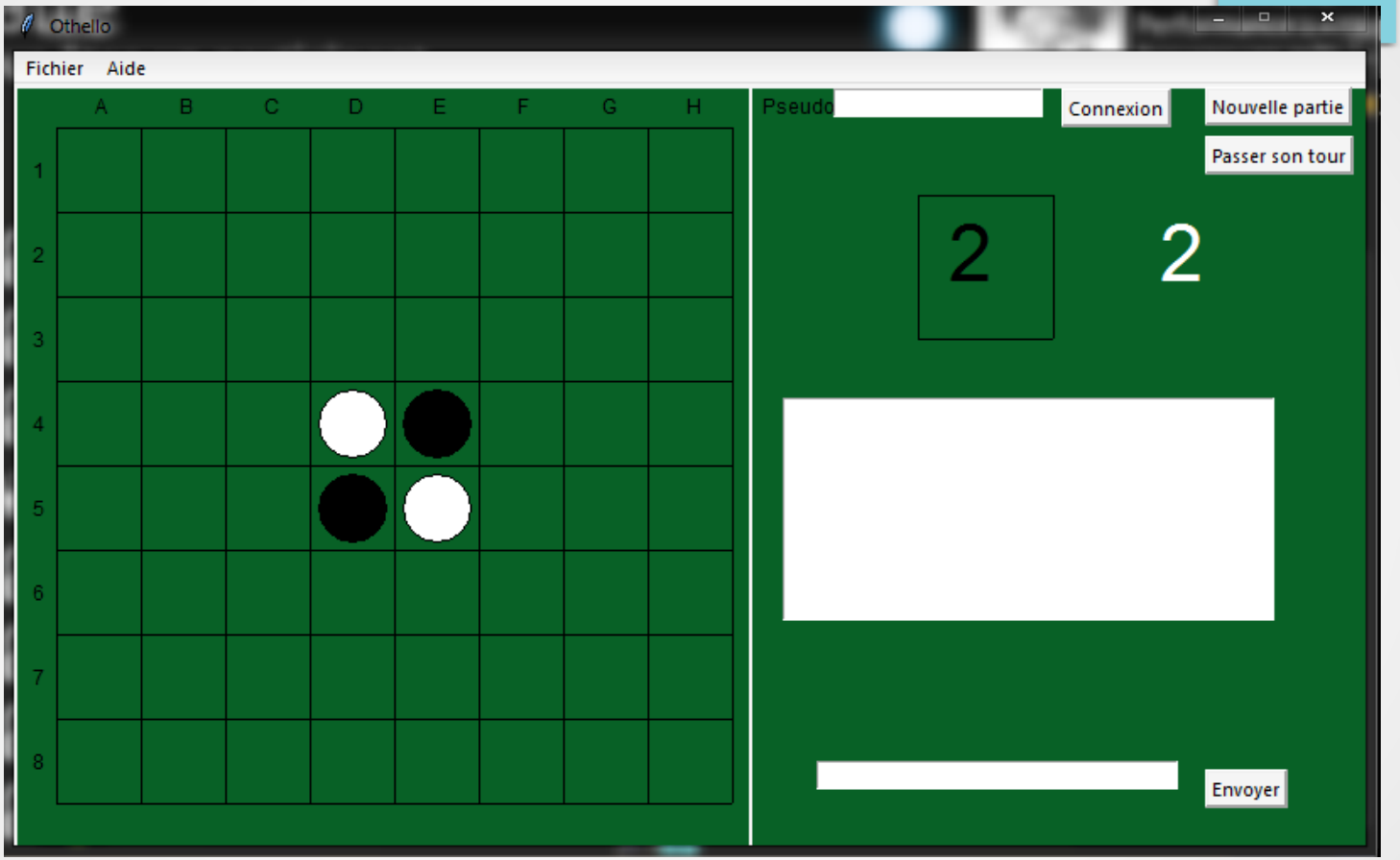
ISN - Projet Othello

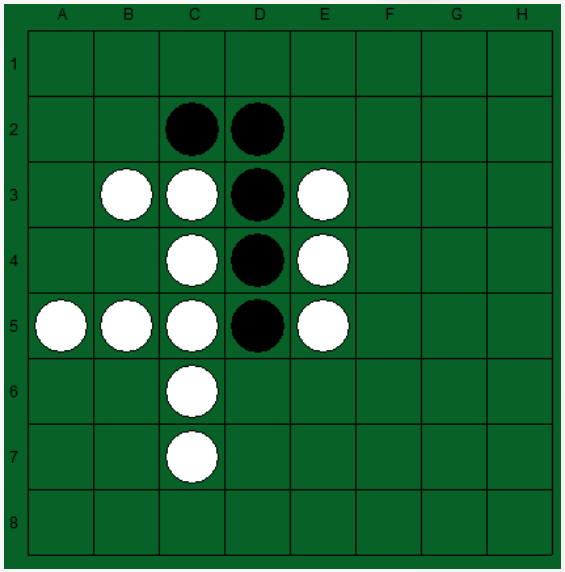


Projet par :

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

Présentation du projet

- Programmation d'un jeu Othello en langage Python.
- Jeu opposant deux humains sur deux ordinateurs différents.
- Respect du jeu de référence :
  - Plateau de jeu identique
  - Respect des règles de placement



Répartition des tâches au sein du groupe

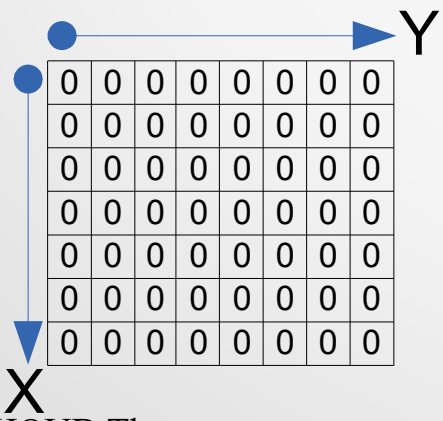
COUCHOUD Thomas : Création de l'algorithme déterminant les impacts des coups joués.
FROGER Olivier : Création de l'interface utilisateur.
JACQUES Johann : Création d'un système permettant la synchronisation entre les deux terminal.

Présentation de l'algorithmique

- Aspect global :
  - Gestion de la grille d'Othello
  - Gestion des différents placements de pions
  - Capacité à donner des informations sur la grille

- Gestion des données :

  Données stockées dans un tableau d'entiers à deux dimensions.



Modification d'une position (x;y)

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

Présentation de l'algorithmique (2)

- Fonctions d'informations :
  - GetColor

```
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]
```

  - GetNumberColor

```
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 demmandee 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
```

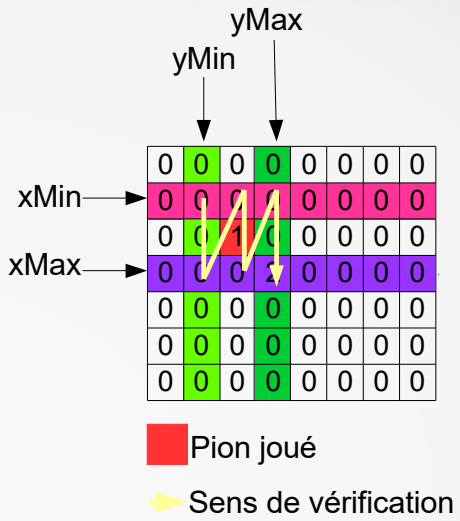
Présentation de l'algorithmique (3)

- Fonctions utilitaires :
  - place

```
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 (vertivale)
    Return:
        0 -> Pas d'errur 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
```

Présentation de l'algorithmique (4)

- Fonctions utilitaires :
  - hasPawnNext

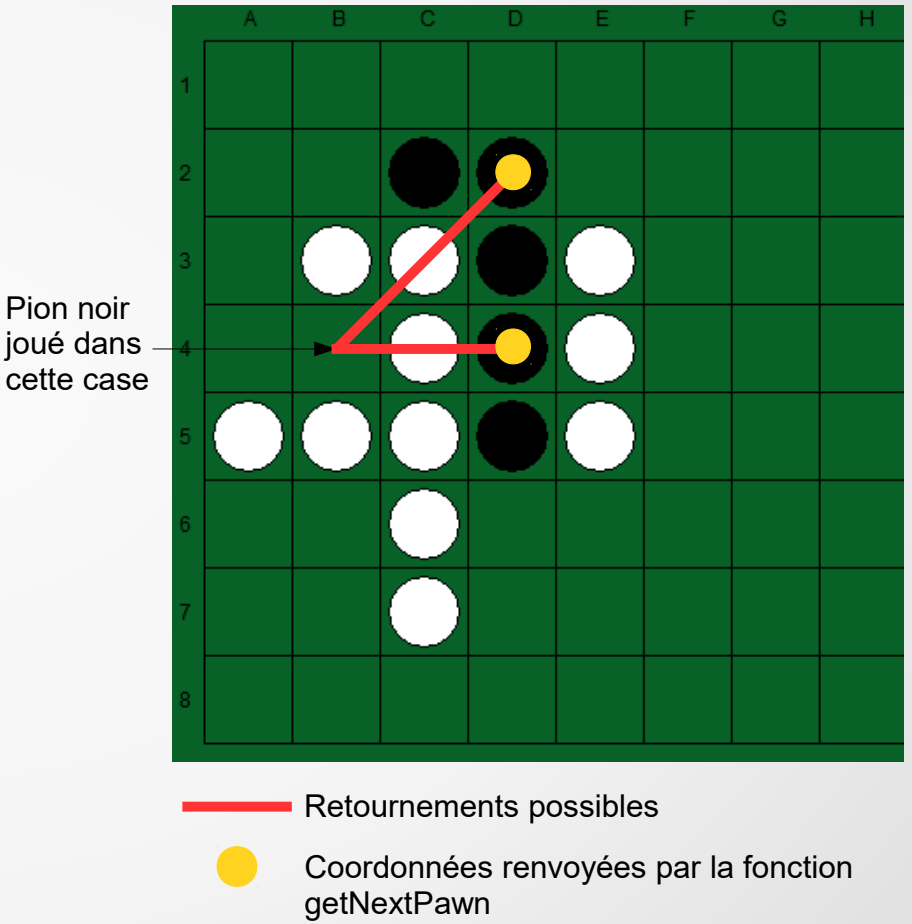



```
def hasPawnNext(color, x, y):
    """
    Permet de savoir si un pion est entoure d'autres pions d'une autre couleur
    Arguments:
        x -> La position x du pion (horizontale)
        y -> La position y du pion (vertivale)
    Return:
        True -> Le pion est entoure d'autres pions
        False -> Le pion n'est pas entoure d'autres pions
    """
    xMin = x - 1 #XMin du carre de 3x3 entourrant le pion
    xMax = x + 1 #XMax du carre de 3x3 entourrant 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 entourrant le pion
    yMax = y + 1 #YMax du carre de 3x3 entourrant 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 entourrant 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
    return False
```

Présentation de l'algorithmique (5)

- Fonctions utilitaires :
  - DetectPawn

```
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
```



Présentation de l'algorithmique (6)

- Fonctions utilitaires :
  - GetNextPawn

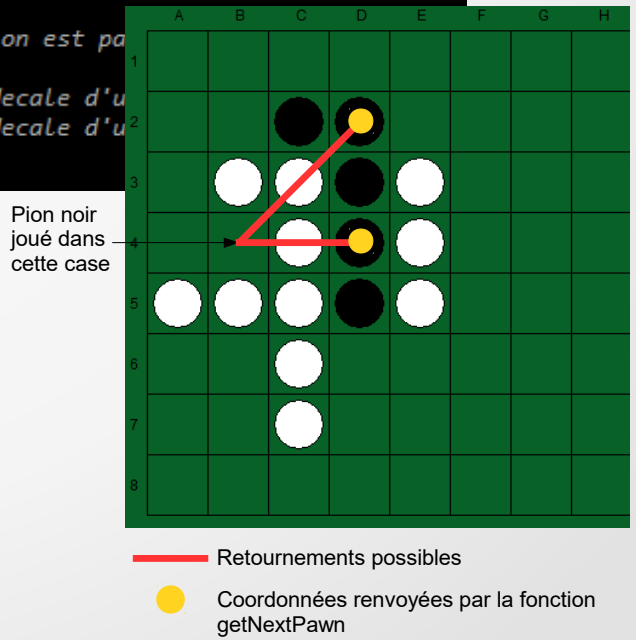
```python
def detectPawn(color, x, y):
    """
    Permet de connaitre les pions de meme couleur qui sont sur les memes lignes
        Arguments:
        color -> La couleur du pion
        x -> La position x du pion (horizontale)
        y -> La position y du pion (verticale)
    Return:
        Une 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 efectue, 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 apsse 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
```


```python
if(directionX == 0 and directionY == 0): return #Aucun decalage efectue, on arrete
for temporaryPos in range(1, 8):
    tempX = int(x + copysign(abs(directionX) * temporaryPos, directionX))
    tempY = int(y + copysign(abs(directionY) * temporaryPos, directionY))
```

Présentation de l'algorithmique (7)

- Fonctions utilitaires :

```
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 coordonnes du pion origine
        coordinateTo -> Les coordonnes 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 pa
        grille[temporaryY][temporaryX] = color #On retourne le pion
        if(deltaX != 0): temporaryX = int(temporaryX + copysign(1, deltaX)) #On se decale d'u
        if(deltaY != 0): temporaryY = int(temporaryY + copysign(1, deltaY)) #On se decale d'u
    grille[coordinateTo[1]][coordinateTo[0]] = color #On retourne le pion
```



Conclusion

- Erreurs commises :
  - Algorithmes ne respectant pas les règles
- Point fort :
  - Partie totalement indépendante donc réutilisable pour un autre projet Othello
- Evolution possibles :
  - Supporter différents tailles de grille