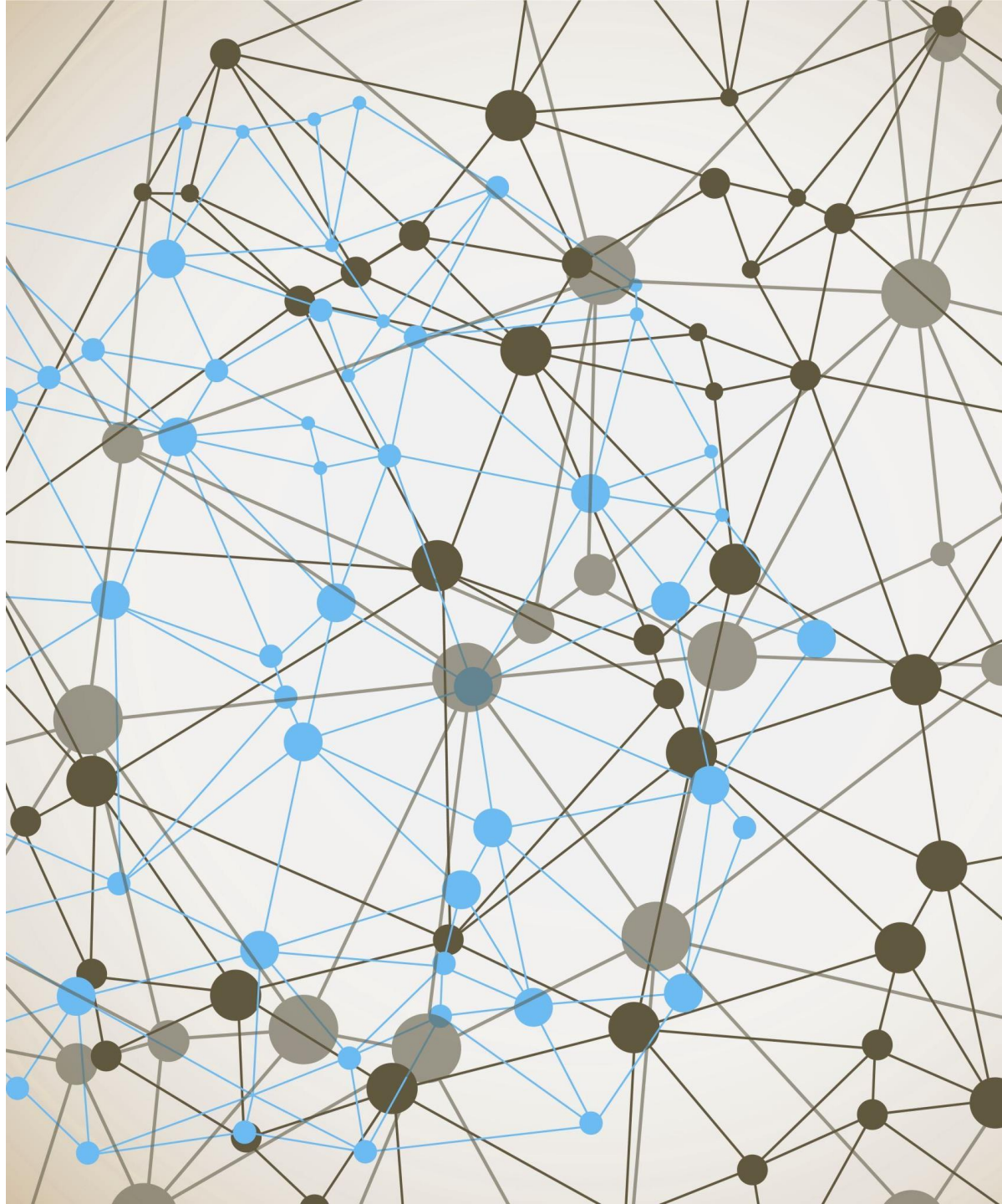
DÉVELOPPEMENT D'UNE IA POUR UN JEU D'ÉCHECS

Ange Herman KOUE-HEMAZRO

Eric NZABA

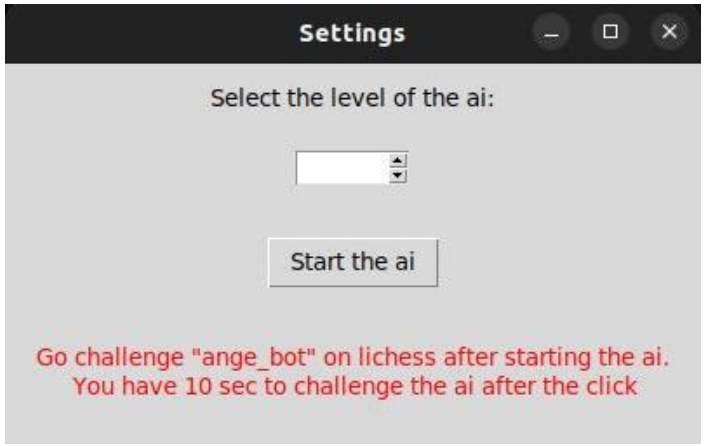


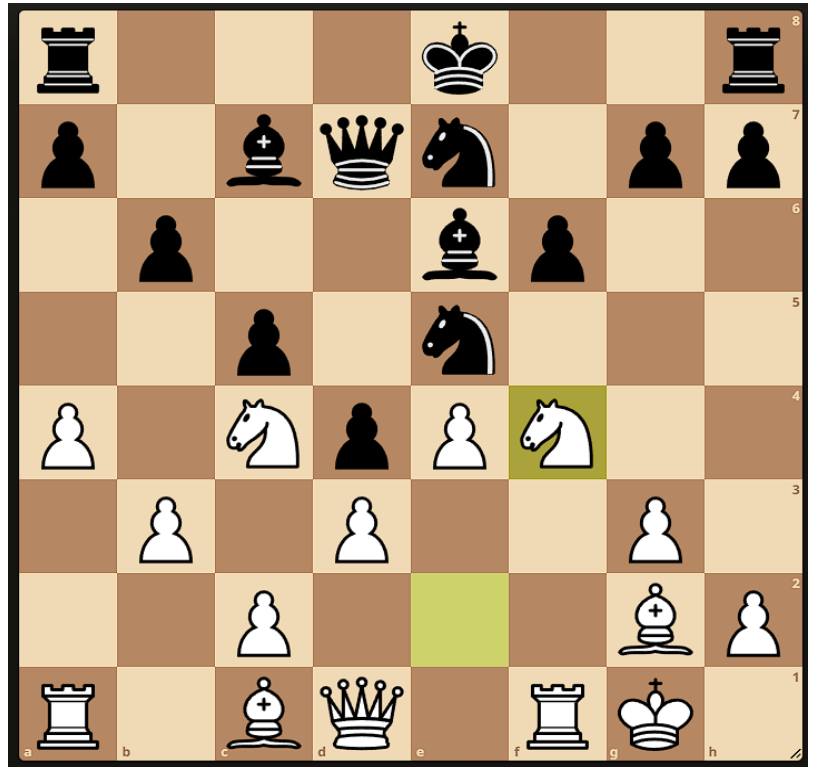
INTRODUCTION

1610

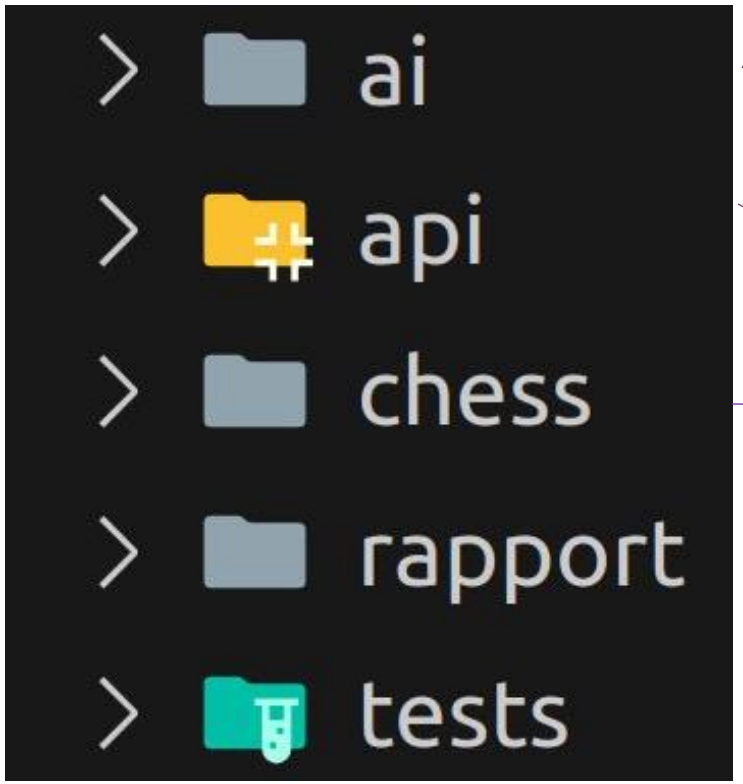
De nos jours …

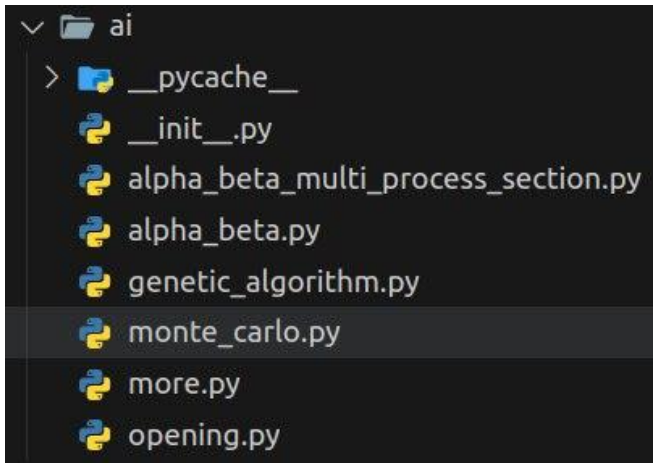
INTRODUCTION/SCENARIO

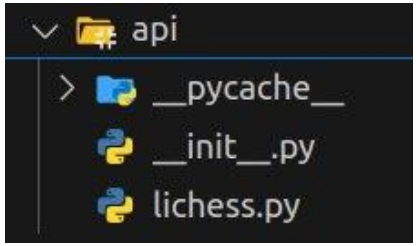


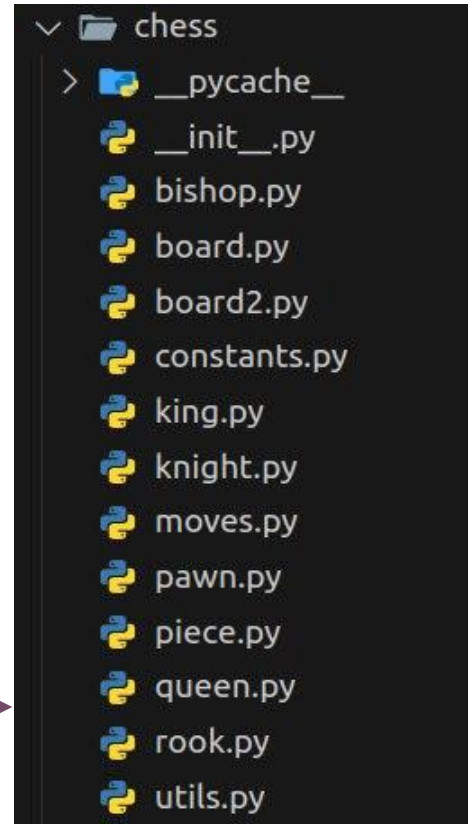


ARCHITECTURE

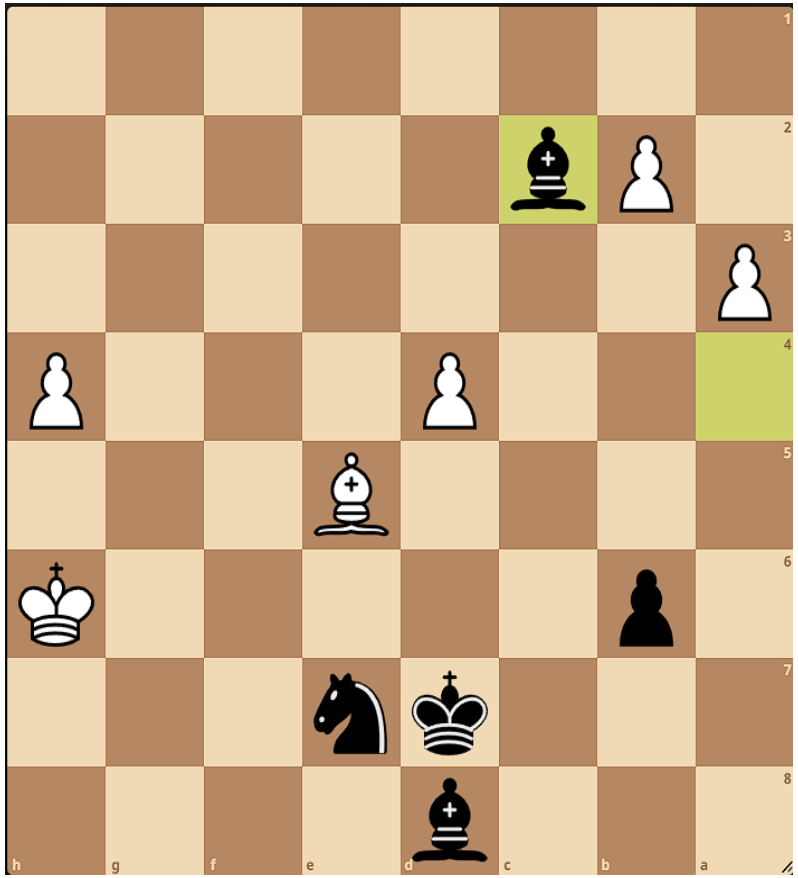








METHODE DE TESTS



```python
def test_situation10():
    # https://lichess.org/JEafAHcd
    plateau = Board2()
    moves = ['e2e4', 'e7e6', 'f1c4', 'd7d5', 'c4b5', 'c7c6', 'b5a4', 'b7b5',
    'a4b3', 'd5e4', 'd1g4', 'f7f5', 'g4h5', 'e8d7', 'h5f7', 'f8e7', 'b3e6',
    'd7c7', 'e6c8', 'd8c8', 'f7g7', 'b8d7', 'g7h8', 'g8f6', 'h8g7', 'c7d6',
    'h2h4', 'b5b4', 'g2g4', 'd7e5', 'f2f4', 'e5g4', 'd2d4', 'f6d5', 'c2c4',
    'g4e3', 'c4d5', 'e3g4', 'a2a4', 'c6d5', 'c1d2', 'd6d7', 'h1h3', 'c8c4',
    'b2b3', 'c4c2', 'd2b4', 'c2c1', 'e1e2', 'c1b2', 'b1d2', 'd7c6', 'b4e7',
    'c6b7', 'e7a3', 'b7c8', 'a3b2', 'h7h5', 'g7g8', 'c8c7', 'g8a8', 'e4e3',
    'a8a7', 'c7c6', 'a7c5', 'c6b7', 'c5d5', 'b7a7', 'd2c4', 'g4f2', 'h3g3',
    'a7b8', 'e2e3', 'f2e4', 'g3g6', 'e4d2', 'e3d2', 'b8c8', 'g6g8', 'c8c7']
```

```python
def test_situation9():
    # https://lichess.org/88oH9ntF
    plateau = Board2()
    plateau.grille = [
        [Rook(BLANC), None, None,  King(BLANC), Queen(BLANC), Bishop(BLANC),Knight
        (BLANC) , Rook(BLANC)],
        [None,Pawn(BLANC),Pawn(BLANC),None,None,None,Pawn(BLANC),Pawn(BLANC)],
        [None,None,Knight(BLANC),None,None,Pawn(BLANC),None,None],
        [Pawn(BLANC),Bishop(NOIR),None,Pawn(BLANC),None,Bishop(BLANC),None,None],
        [None,None,None,Pawn(NOIR),Pawn(NOIR),Pawn(BLANC),None,None],
        [Knight(NOIR),None,None,None,None,Knight(NOIR),None,None],
        [Pawn(NOIR),Pawn(NOIR),Pawn(NOIR),None,None,Pawn(NOIR),Pawn(NOIR),Pawn(NOIR)],
        [Rook(NOIR),None, None,  King(NOIR), Queen(NOIR), None, None, Rook(NOIR)]
    ]
```

DIFFICULTÉS ET RÉSOLUTIONS

- Dans l'ensemble c'est la fiabilité de l'IA (la précision, vitesse, sans bugs de mouvement etc …)
- Implémentation du plateau de jeu
- Implémentation des IA
- Evaluation du plateau
- Bugs des mouvements (gérer avec sérialisation)
- Vitesses des IA

GESTION DU PROJET

- Temps de développement
- Répartition des tâches

PROGRAMMATION

```python
def getKingSurrondings(self,piece):

    dangerousCoordinates = []

    straight = straightPathsFromPiece(piece,self.grille,HEIGHT,WIDTH)
    diagonals = diagonalPathsFromPiece(piece,self.grille,WIDTH)

    straightLineEnemies = {TOUR,DAME}
    diagonalLineEnemies = {FOU,DAME}
    knightEnemy = CAVALIER

    for i in range(len(straight)):

        """ On vérifie bien que la dernière pièce dans la direction choisi est une pièce ennemie
        ayant la capacité de mettre en danger le roi,
        On vérifie aussi qu'entre ses deux pièces il n'existe pas une pièce allié """
        retrievePiece = getPiece(self.grille,straight[i][len(straight[i]) - 1])
        if retrievePiece != None:
            if (retrievePiece.name in straightLineEnemies) and retrievePiece.color != piece.color :
                dangerousCoordinates.append(("STRAIGHT",straight[i][1:]))


    for i in range(len(diagonals)):

        retrievePiece = getPiece(self.grille,diagonals[i][len(diagonals[i]) - 1])
        if retrievePiece != None:
            if retrievePiece.name  in diagonalLineEnemies and retrievePiece.color != piece.color :
                dangerousCoordinates.append(("DIAGONAL",diagonals[i][1:]))
            if retrievePiece.name == PION and retrievePiece.color != piece.color and len(diagonals[i]) == 2:
                dangerousCoordinates.append(("DIAGONAL",diagonals[i][1:]))

    simulated_Knight = Knight(piece.color)
    simulated_Knight.setCoordinates(piece.coordinates)
    knights = simulated_Knight.knight_movement(self.grille)

    threateningKnights = [knights[i][1] for i in range(len(knights)) if checkPieceName(self.grille,knights[i][1],knightEnemy)]

    if(len(threateningKnights)>0):
        dangerousCoordinates.append(("L",threateningKnights))

    return dangerousCoordinates
```

```python
def getKingProtectionList(self,kingSurroundings):
    protectList = []
    threatenedPathsToRemove = []

    for i in range(len(kingSurroundings)):

        if(kingSurroundings[i][0] == "STRAIGHT" or kingSurroundings[i][0] == "DIAGONAL"):

            length = len(kingSurroundings[i][1]) - 1
            pieces = self.getPieceCoordinatesInBetweenPath(kingSurroundings[i][1][:length])

            if len(pieces) > 0:
                threatenedPathsToRemove.append(i)
            if len(pieces) == 1:
                protectList += pieces


    kingSurroundings2 = [kingSurroundings[i] for i in range(len(kingSurroundings)) if i in threatenedPathsToRemove]

    for i in range(len(kingSurroundings2)):
        kingSurroundings.remove(kingSurroundings2[i])

    return protectList
```

```python
    if pieces[i].name == ROI:
        kingSurroundings = self.getKingSurrondings(pieces[i])
        kingPosition = pieces[i].coordinates
        if pieces[i].moveCount == 0:
            pMoves += self.checkRoqueAvailability(kingPosition)


# Si le roi est directement menacé par une pièce on retourne uniquement les mouvement permettant de le protéger
kingSurroundingsFlattened = [kingSurroundings[i][1][j] for i in range(len(kingSurroundings)) for j in range(len(kingSurroundings[i][1]))]
protectionList = self.getKingProtectionList(kingSurroundings)
```

CONCLUSION

- Amélioration de l'implémentation du plateau
- Table de transposition
- Affrontement de deux IA
- Outil d'analyse de fiabilité d'algorithme