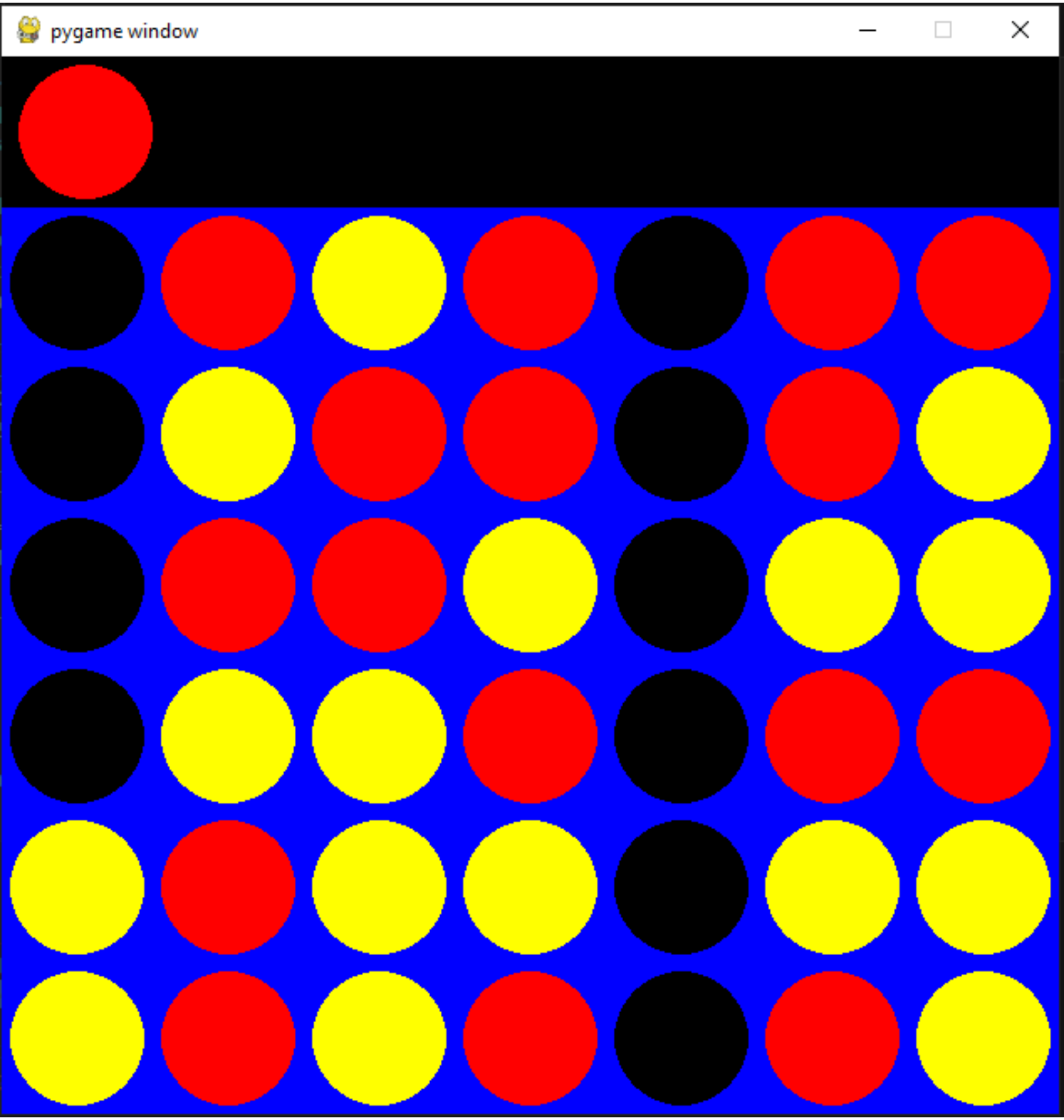
Jeu du Puissance 4



I – Introduction et contexte

a – Brève introduction au jeu de Puissance 4

Puissance 4 est un jeu de stratégie combinatoire abstrait, commercialisé pour la première fois en 1974 par la Milton Bradley Company et détenue depuis 1984 par la société Hasbro.

Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Tour à tour, les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans ladite colonne à la suite de quoi c'est à l'adversaire de jouer.

Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.

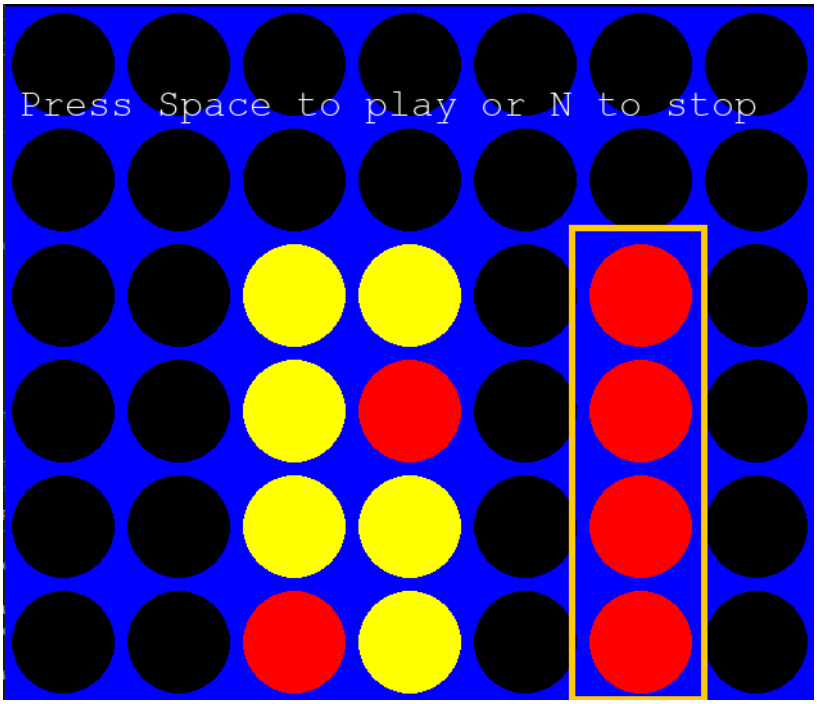


Figure 1: Ici le joueur utilisant les jetons rouge à gagner (rectangle orange)

b - Présentation des technologies utilisées (Python, Pygame)

Python:

Créé par Guido van Rossum et publié pour la première fois en 1991 Python est un langage de programmation puissant et facile à apprendre. Il dispose de structures de données de haut niveau et permet une approche simple mais efficace de la programmation orientée objet.

Parce que sa syntaxe est élégante, que son typage est dynamique et qu'il est interprété (ce qui signifie que le code source est directement exécuté par un interpréteur sans nécessiter de compilation préalable), Python est un langage idéal pour l'écriture de scripts et le développement rapide d'applications dans divers domaines tels que :

Le développement web, l'analyse de données, l'intelligence artificielle, l'automatisation de tâches, les applications scientifiques, et bien plus encore.

```python
# Exemple simple en Python
def dire_bonjour(nom):
    print("Bonjour, " + nom + " !")

# Appel de la fonction
dire_bonjour("Alice")
```

Figure 2: Cet exemple illustre la simplicité et la clarté du langage Python. Il définit une fonction dire_bonjour qui prend un paramètre nom et imprime un message de salutation. Ensuite, la fonction est appelée avec le nom "Alice".

Pygame:

Pygame est une bibliothèque open source dédiée au développement de jeux vidéo et d'applications multimédias en Python. Elle repose sur la bibliothèque Simple DirectMedia Layer (SDL) et offre un ensemble d'outils et de fonctionnalités pour faciliter la création de jeux et d'applications graphiques interactives.
Pygame prend en charge la gestion des événements, les graphiques, le son, les images, et offre des fonctionnalités pour la création d'interfaces utilisateur graphiques (GUI) interactives.

```python
import pygame
import sys

# Initialisation de Pygame
pygame.init()

# Définition de la fenêtre
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Premier Jeu Pygame")

# Boucle Principale
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    # Dessiner un fond bleu
    screen.fill((0, 0, 255))

    # Mettre à jour l'affichage
    pygame.display.flip()
```

Figure 3: Cet exemple crée une fenêtre Pygame basique et utilise une boucle principale pour détecter les événements (comme la fermeture de la fenêtre) et pour redessiner continuellement le fond en bleu.

II – Conception du jeu

Description de l'interface utilisateur et logique du jeu

L'interface affiche un plateau de jeu rectangulaire avec 6 rangées et 7 colonnes, formant une grille de 6x7. Les emplacements vides sont initialement représentés par des cercles vides.

Deux couleurs distinctes sont utilisées pour représenter les jetons des deux joueurs. Par exemple, des jetons rouges pour le joueur 1 (humain) et des jetons jaunes pour le joueur 2 (IA ou autre joueur)

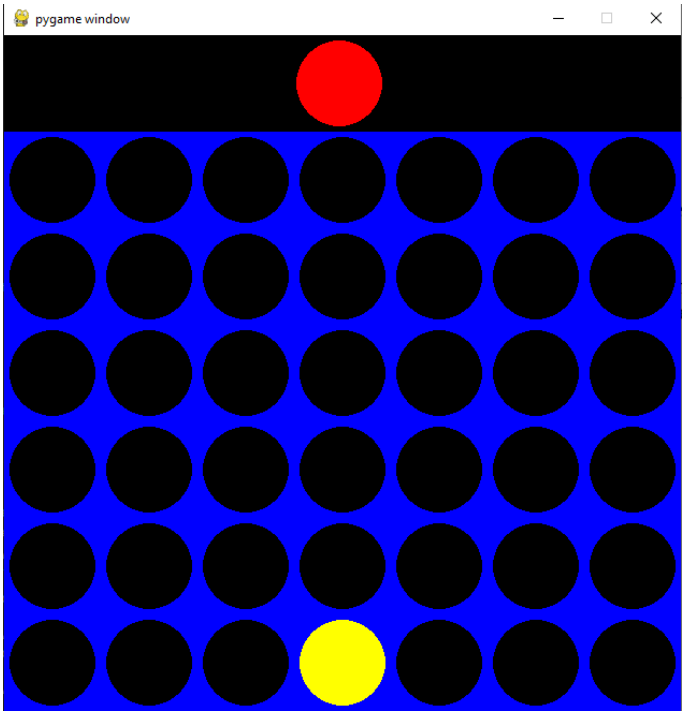


Figure 4: Ici le joueur 2 qui utilise les jetons jaune à jouer son premier jeton, le joueur 1 peut à son tour placer son jeton rouge dans la colone de son choix

Un indicateur visuel montre quel joueur est actuellement en train de jouer (Fig.4). La direction du jeton suit la direction de la souris, ainsi lors du clique gauche de la souris le joueur 1 (jeton rouge) dépose son jeton sur la colone qu’il a choisi.

En cas de victoire, l'interface affiche un message de félicitations, indiquant quel joueur a gagné. Par exemple, "Player 1 win!".

Après la fin d'une partie, l'interface affiche un message invitant le joueur à rejouer. Les options comme "Press Space to play" et "N to quit" sont proposées, permettant au joueur de lancer une nouvelle partie ou de quitter le jeu.

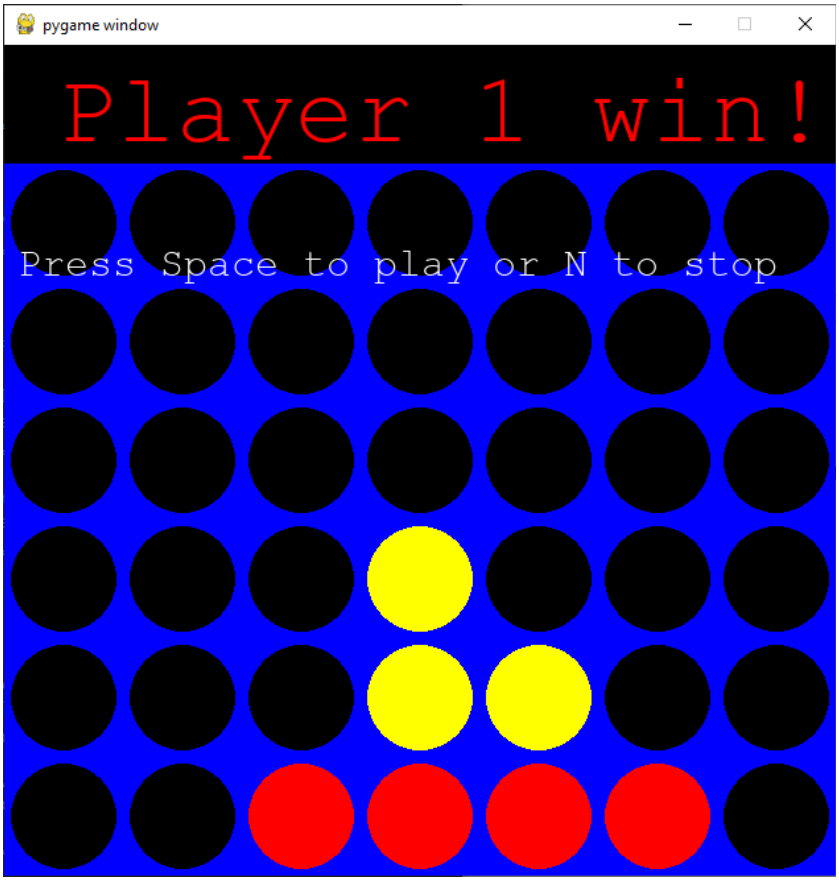
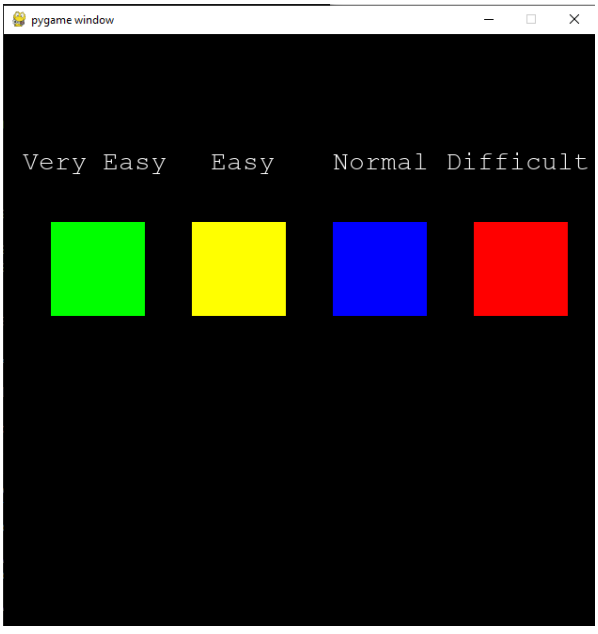


Figure 5: Après la victoire du joueur 1 (jeton rouge) l'interface propose au joueur de rejouer ou quittter

Le jeu propose différents niveaux de difficulté avant de commencer chaque partie. En fonction du choix du joueur, la difficulté sera aisé (Fig.5) ou difficile.



III – Développement

a – Interface utilisateur et logique de jeu

Création de la fenêtre de jeu:

Pour créé de la fenêtre de jeu on commence par importer le module pygame: `import pygame`

Ensuite on initialise pygame pour l'utiliser: `pygame.init()`

On définit les dimensions de la fenêtre:

```
ROW_COUNT = 6 # Nombre de rangée
COLUMN_COUNT = 7 # Nombre de colone
SQUARESIZE = 90 # Initialise et assigne une valeur pour la taille qui sera multiplié
width = COLUMN_COUNT * SQUARESIZE # Assigne la largeur de la fenêtre par le nombre de colone * la taille
height = (ROW_COUNT+1) * SQUARESIZE # Assigne la hauteur de la même façcon, en ajoutant +1 pour la rangée permettant au joueur de voir son jeton avant de le placer
```

Et on créer la fenêtre avec la largeur et la hauteur définie:

```
size = (width, height)
screen = pygame.display.set_mode(size)
```

Pour maintenir la fenêtre on applique une boucle principale (Event Loop):

```
game_over = False
while not game_over: # Tant qu'il n'y a pas de game over
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
```

Affichage du plateau de jeu et des jetons et Gestion des entrées utilisateur :

D'abord on créé le plateau de jeu avec un tableau bidimensionnel (board) de largeur ROW_COUNT par la longueur COLUMN_COUNT initialisé à zéro, représentant les emplacements vides:

```
def created_board(): # on déclare la fonction
    board = np.zeros((ROW_COUNT, COLUMN_COUNT))# la variable board étant initialisé avec un tableau, sa première valeur sera [0]
    return board # A la fin de l'appel de la fonction board est retourné
```

À l'aide de la fonction draw_board(board), une boucle for traverse chaque cellule du plateau.Un rectangle bleu est dessiné pour chaque cellule, formant ainsi le plateau.Un cercle noir est dessiné au centre de chaque cellule, représentant un emplacement vide.

Puis avec le même principe on change la couleur des cercles noir en cercle rouge si le jeton posé correspond à 1 (joueur 1) et jaune si il correspond à 2 (joueur 2).

On termine la fonction avec une actualisation de la fenêtre à chaque appel de la fonction pour afficher les changements du plateau:

```
def draw_board(board):
  for c in range(COLUMN_COUNT):
     for r in range(ROW_COUNT):

                pygame.draw.rect(screen, BLUE, (c*SQUARESIZE, r*SQUARESIZE+SQUARESIZE, SQUARESIZE,  SQUARESIZE), width=0)


                pygame.draw.circle(screen, BLACK, (int(c*SQUARESIZE+SQUARESIZE/2),
                int(r*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)), RADIUS)

   for c in range(COLUMN_COUNT):
      for r in range(ROW_COUNT):
                if board[r][c] == 1:
                   pygame.draw.circle(screen, RED, (int(c*SQUARESIZE+SQUARESIZE/2), height-
                int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)

                elif board[r][c] == 2:
                   pygame.draw.circle(screen, YELLOW, (int(c*SQUARESIZE+SQUARESIZE/2), height-
                int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)

  pygame.display.update()
```

La fonction pygame.draw.circle est utilisée pour dessiner les jetons à l'emplacement approprié.

Lorsqu'un joueur dépose un jeton, la fonction drop_piece(board, row, col, piece) place le jeton dans la cellule spécifiée par row et col:

```
def drop_piece(board, row, col, piece):
   board[row][col] = piece
```

La fonction winning_move permet de vérifié en retournant True si un joueur valide 4 emplacements de jetons:

```
def winning_move(board, piece):
   # Vérifie les positions de 4 jetons horizontalement
   for c in range(COLUMN_COUNT-3):
     for r in range(ROW_COUNT):
                if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] == piece:
                   return True

   # Vérifie les positions de 4 jetons verticalement
   for c in range(COLUMN_COUNT):
      for r in range(ROW_COUNT-3):
                if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3][c] == piece:
                   return True

   #Vérifie les position diagonals positive
   for c in range(COLUMN_COUNT-3):
     for r in range(ROW_COUNT-3):
           if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and board[r+3][c+3] == piece:
              return True

   # Vérifie les position diagonals négative
   for c in range(COLUMN_COUNT-3):
      for r in range(3, ROW_COUNT):
           if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and board[r-3][c+3] == piece:
              return True
```

Lorsqu'un joueur remporte la partie (fonction winning_move), un message est affiché en utilisant pygame.font.render indiquant le joueur gagnant:

```
if winning_move(board, 1):
                    label = myfont.render("Player 1 win!", 1, RED)
                     screen.blit(label, (40, 10))
                     game_over = True
```

La condition d'affichage de victoire est intégré à la boucle principale (while not game_over) qui gère les événements de la fenêtre Pygame, tels que les clics de souris et les mouvements de souris. Lorsqu'un clic est détecté, la position de la souris est utilisée pour déterminer la colonne où le joueur souhaite déposer son jeton. Le jeu alterne entre les joueurs (tour 0 pour le joueur 1, tour 1 pour le joueur 2) à chaque clic:

```
turn = 0
while not game_over:

   # Gére les entrées des joueurs
    for event in pygame.event.get():
      if event.type == pygame.QUIT:
            sys.exit()

             # Gére l'entrée de la position de la souris
      if event.type == pygame.MOUSEMOTION:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            posx = event.pos[0]
            if turn == 0:
              pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE/2)), RADIUS)
            else:
              pygame.draw.circle(screen, YELLOW, (posx, int(SQUARESIZE/2)), RADIUS)
      pygame.display.update()

            # Gére le clique de la souris
      if event.type == pygame.MOUSEBUTTONDOWN:
         pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))

         #Si le tour est égal à 0 alors le joueur 1 joue
         if turn == 0:
         posx = event.pos[0]
         col = int(math.floor(posx/SQUARESIZE))
            if is_valid_location(board, col):
              row = get_next_open_row(board, col)
              drop_piece(board, row, col, 1)
              if winning_move(board, 1):
                   label = myfont.render("Player 1 win!", 1, RED)
                   screen.blit(label, (40, 10))
                   game_over = True
        #Sinon c'est le tour du joueur 2
        else:
        posx = event.pos[0]
           col = int(math.floor(posx/SQUARESIZE))
           if is_valid_location(board, col):
             row = get_next_open_row(board, col)
             drop_piece(board, row, col, 2)
             if winning_move(board, 2):
                  label = myfont.render("Player 2 win!", 1, YELLOW)
                  screen.blit(label, (40, 10))
                  game_over = True
         draw_board(board)
         turn +=1
         turn = turn % 2
       if game_over:
          pygame.time.wait(3000)
```

b - Implémentation d'une IA basique

La fonction pick_best_move(board, piece) permet de choisir la meilleur colonne pour l'IA. Elle évalue les différentes positions possibles en utilisant la fonction score_position et sélectionne la colonne avec le meilleur score. score_postion évalue la position actuelle du plateau pour un joueur donné (piece). Elle attribue des scores en fonction des configurations de jetons présentes sur le plateau. L'algorithme explore les mouvements possibles et attribue des scores en fonction de la position des jetons. La colone central est préféré de façon à simuler une stratégie connue du Puissance 4.

```
def score_position(board, piece):
    score = 0
    # Score colone central
    center_array = [int(i) for i in list(board[:, COLUMN_COUNT//2])]
    center_count = center_array.count(piece)
    score += center_count * 6
    # Score horizontal
    for r in range(ROW_COUNT):
        row_array = [int(i) for i in list(board[r,:])]
        for c in range(COLUMN_COUNT-3):
            window = row_array[c:c+WINDOW_LENGTH]
            score += evaluate_window(window, piece)
    # Score vertical
    for c in range(COLUMN_COUNT):
        col_array = [int(i) for i in list(board[:,c])]
        for r in range(ROW_COUNT-3):
            window = col_array[r:r+WINDOW_LENGTH]
            score += evaluate_window(window, piece)
    # Score diagonal positive
    for r in range(ROW_COUNT-3):
      for c in range(COLUMN_COUNT-3):
            window = [board[r+i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)
    # Score diagonal negative
    for r in range(ROW_COUNT-3):
      for c in range(COLUMN_COUNT-3):
            window = [board[r+3-i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)

def pick_best_move(board, piece):
    valid_location = get_valid_location(board)
    best_score = -1000
    best_col = random.choice(valid_location)
    for col in valid_location:
        row = get_next_open_row(board, col)
        temp_board = board.copy()
        drop_piece(temp_board, row, col, piece)
        score = score_position(temp_board, piece)
        if score > best_score:
            best_score = score
            best_col = col
    return best_col
```

La fonction evaluate_window(window, piece) attribue des scores aux fenêtres de jetons dans une ligne, colonne ou diagonale spécifique. Elle contribue à l'évaluation globale de la position.

```
def evaluate_window(window, piece):
    score = 0
    opp_piece = PLAYER_PIECE
    if piece == PLAYER_PIECE:
    opp_piece = AI_PIECE

    if window.count(piece) == 4:
        score += 100
    elif window.count(piece) == 3 and window.count(EMPTY) == 1:
        score += 10
    elif window.count(piece) == 2 and window.count(EMPTY) == 2:
        score += 10

    if window.count(opp_piece) == 3 and window.count(EMPTY) == 1:
        score -= 80

    return score
```

La boucle principal étant modifié. Lorsque c'est le tour de l'IA (turn == AI), la fonction pick_best_move est appelée pour choisir la meilleure colonne, puis le jeton est déposé sur le plateau.

```
    if turn == AI and not game_over:
        col = pick_best_move(board, AI_PIECE)
        if is_valid_location(board, col):
            pygame.time.wait(1000)
            row = get_next_open_row(board, col)
            drop_piece(board, row, col, AI_PIECE)
            if winning_move(board, AI_PIECE):
                label = myfont.render("Player 2 win!", 1, YELLOW)
                screen.blit(label, (40, 10))
                game_over = True
            draw_board(board)
            turn +=1
            turn = turn % 2
```

c - Amélioration de l'IA avec l’algorithme Minimax

Implémentation l'algorithme Minimax avec la fonction minimax(board, depth, maximizingPlayer). Il explore les mouvements possibles jusqu'à une certaine profondeur, évalue les positions résultantes et attribue des scores. A chaque appel de la fonction minimax, la fonction is_terminal_node(board) vérifie si le jeu est terminé en raison d'une victoire ou si toutes les colonnes sont remplies:

```
def is_terminal_node(board):# Si Vrai, c’est un noeud de fin de jeu
    return winning_move(board, PLAYER_PIECE) or winning_move(board, AI_PIECE) or
        len(get_valid_location(board)) == 0
# Algorithme minimax, profondeur de branche definie par la variable depth
def minimax(board, depth, maximizingPlayer):
   valid_location = get_valid_location(board) # Assigne les corrects localisation de jeu possible
   is_terminal = is_terminal_node(board) # Si faux, continue
   if depth == 0 or is_terminal:# Stop la fonction recursive de prévision
      if is_terminal: # Si c’est un noeud de fin de jeu
         if winning_move(board, AI_PIECE):
            return (None, 1000000000000000)
         elif winning_move(board, PLAYER_PIECE):
            return (None, -100000000000000)
         else:# Match nulle
            return (None, 0)
      else:
         return (None, score_position(board, AI_PIECE))
# Partie où l’on cherche à maximiser les points du joueur IA
   if maximizingPlayer:
      value = -math.inf # Une valeur infinie permet de sécuriser l’assination de la futur valeur
      column = random.choice(valid_location)
      for col in valid_location:
         row = get_next_open_row(board, col)
         b_copy = board.copy()
         drop_piece(b_copy, row, col, AI_PIECE)
         new_score = minimax(b_copy, depth-1, False)[1] # Retourne le meilleur score de
l’appel récursive minimax
         if new_score > value:
            value = new_score
            column = col
      return column, value # Retourne la meilleur valeur avec sa colone
   else: # Partie où l’on cherche à minimiser les points du joueur humain
      value = math.inf
      column = random.choice(valid_location)
      for col in valid_location:
         row = get_next_open_row(board, col)
         b_copy = board.copy()
         drop_piece(b_copy, row, col, PLAYER_PIECE)
         new_score = minimax(b_copy, depth-1, True)[1]
         if new_score < value:
            value = new_score
            column = col
     return column, value
```

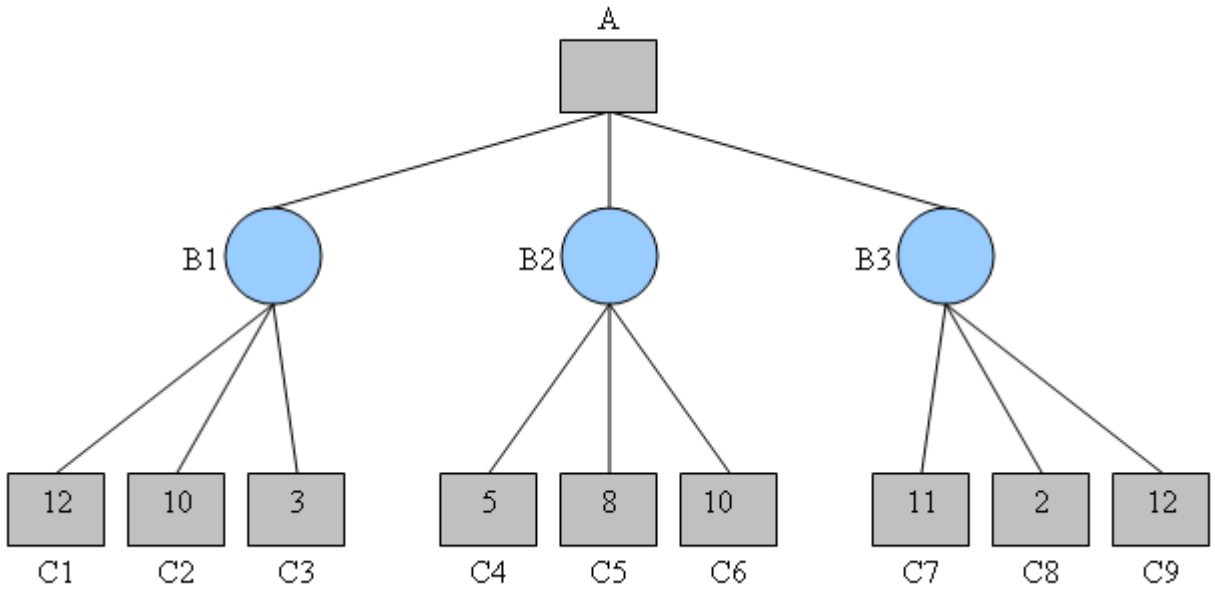
Dans la boucle principale, l’on assigne aux variables col et minimax_score le résultat de l’appel de la fonction minimax:

```
if turn == AI and not game_over:
    col, minimax_score = minimax(board, depth, True)
```

Le principe de l’algorithme Minimax:

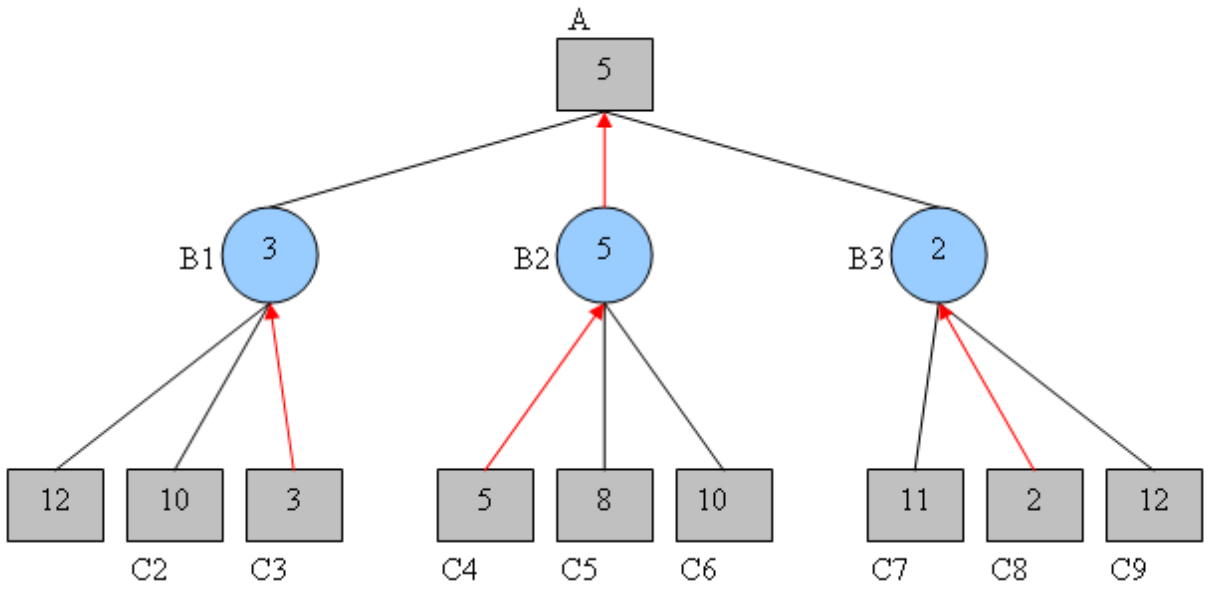
L'algorithme Minimax est un algorithme de recherche de décision utilisé dans les jeux à deux joueurs avec une alternance de tours, tels que les échecs, le Puissance 4, le tic-tac-toe, etc. L'objectif principal de l'algorithme Minimax est de trouver la meilleure stratégie pour un joueur, en supposant que l'adversaire joue de manière optimale. Il explore l'arbre de jeu possible jusqu'à une certaine profondeur, attribue des valeurs aux positions finales et remonte ensuite ces valeurs pour déterminer le meilleur mouvement à un niveau donné.

L'algorithme représente le jeu sous forme d'un arbre, où chaque nœud représente un état possible du jeu. Les niveaux alternent entre les tours du joueur et de l'adversaire. Chaque feuille de l'arbre est évaluée en fonction de la position du jeu à ce stade. Dans le contexte du Puissance 4, une évaluation positive peut être attribuée à une position favorable pour l'IA (alignement de pièces) et une évaluation négative pour une position favorable au joueur humain.



Les valeurs évaluées remontent dans l'arbre. À chaque nœud, le joueur maximisant cherche à maximiser la valeur (score), tandis que le joueur minimisant cherche minimiser la valeur. Les nœuds à un niveau impair (tour du joueur) choisissent le maximum des valeurs de leurs nœuds fils. Les nœuds à un niveau pair (tour de l'adversaire) choisissent le minimum des valeurs de leurs nœuds fils.

Au niveau racine de l'arbre (le premier coup du joueur), l'algorithme choisit le mouvement qui maximise la valeur. À chaque niveau suivant, il sélectionne le mouvement qui minimise la valeur, en supposant que l'adversaire joue de manière optimale. Cela donne le meilleur coup possible pour le joueur, compte tenu de la stratégie optimale de l'adversaire.



Le pseudocode de l'algorithme minimax de profondeur limitée est présenté ci-dessous:

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
    return value

(* Initial call *)
minimax(origin, depth, TRUE
```

IV – Conclusion et Perspectives

Voici les principales étapes et réalisation personnel durant ce projet

Développement du Jeu Puissance 4 :

Mise en œuvre complète du jeu Puissance 4 en utilisant la bibliothèque Pygame.

Création d'une interface utilisateur interactive pour permettre aux joueurs humains de jouer en utilisant la souris.

Gestion de l'affichage graphique du plateau de jeu et des pièces des joueurs.

Intégration de l'IA avec Minimax :

Implémentation de l'algorithme Minimax pour permettre à l'ordinateur de jouer en tant qu'adversaire dans le Puissance 4.

Utilisation de la recherche Minimax pour évaluer les différentes positions possibles et choisir le meilleur mouvement à chaque tour.

Intégration d'une fonction d'évaluation pour attribuer des scores aux positions en fonction de la probabilité de victoire.

Niveaux de Difficulté :

Ajout d'une fonctionnalité de niveaux de difficulté permettant aux joueurs de choisir le niveau de compétence de l'IA.

Mise en place d'une interface utilisateur permettant de sélectionner la difficulté avant de commencer une nouvelle partie.

Gestion de la Fin de Partie :

Mise en place de mécanismes pour détecter la fin du jeu, qu'il s'agisse d'une victoire, d'une défaite ou d'un match nul.

Affichage des résultats à l'écran, indiquant quel joueur a gagné ou si la partie s'est terminée par un match nul.

Répétition du Jeu et Interface Utilisateur :

Ajout d'une fonctionnalité de redémarrage du jeu après la fin d'une partie.

Affichage d'un message invitant le joueur à rejouer ou à quitter, avec des commandes clavier pour prendre une décision.

Expérience Utilisateur Améliorée :

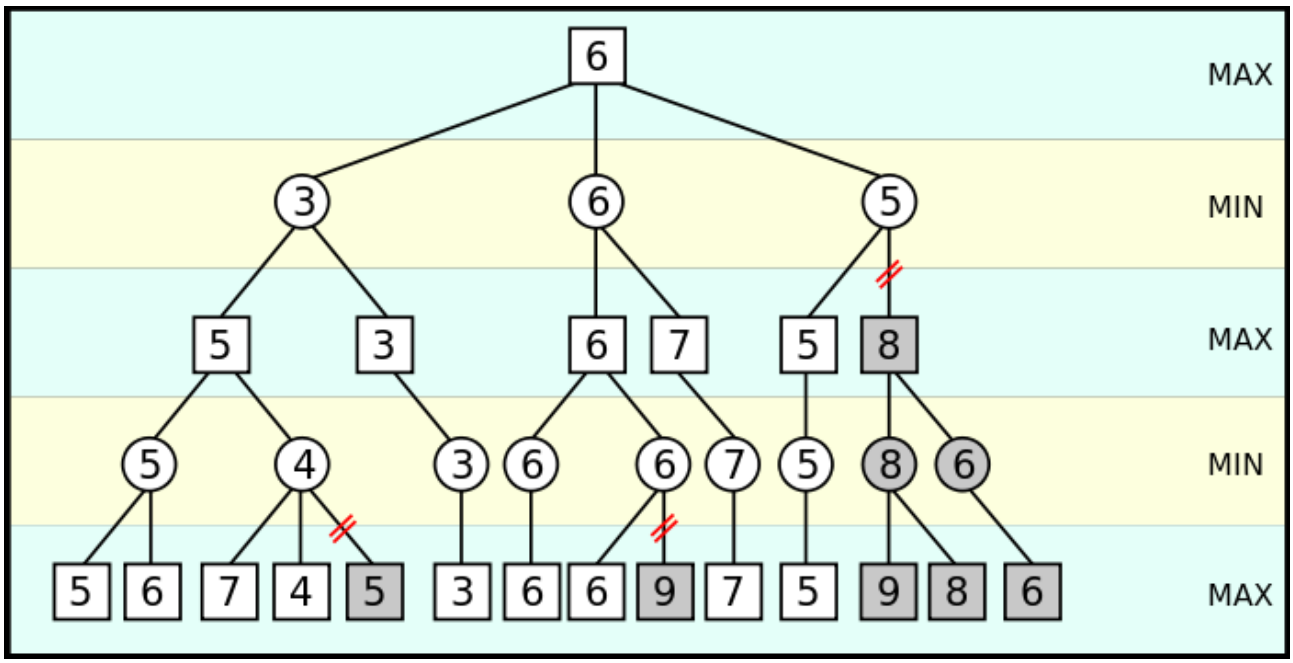
Création d'une interface utilisateur claire et intuitive, avec des indications visuelles sur le déroulement du jeu.

Utilisation de polices de caractères, de couleurs et de formes pour rendre l'expérience utilisateur agréable et compréhensible.

Ce projet a permis d'acquérir des compétences pratiques en programmation de jeux avec Pygame, ainsi que la compréhension et l'implémentation d'un algorithme d'intelligence artificielle classique (Minimax) pour prendre des décisions dans un contexte de jeu. L'intégration de fonctionnalités telles que les niveaux de difficulté et la gestion de la fin de partie améliore l'expérience globale du joueur.

Amélioration possible

L’amélioration de minimax par élagage Alpha-Béta permettrai d'optimiser grandement l'algorithme minimax sans en modifier le résultat. Pour cela, il ne réaliserait qu'une exploration partielle de l'arbre. Lors de l'exploration, il n'est pas nécessaire d'examiner les sous-arbres qui conduisent à des configurations dont la valeur ne contribuera pas au calcul du gain à la racine de l'arbre. Dit autrement, l'élagage Alpha-Béta n'évalue pas des noeuds dont on peut penser, si la fonction d'évaluation est à peu près correcte, que leur qualité sera inférieure à celle d'un nœud déjà évalué.



Autres améliorations possible, celle de l’interface utilisateur par l’ajout d’un scoring, d’un temps de jeu et d’un menu proposant soit de jouer contre un autre joueur humain soit contre l’IA ou de l’ajout de musique.

V – Annexes et sources

Codes sources du fichier joueur vs joueur:

```
import numpy as np
import pygame
import sys
import math

#Import des bibliothèques nécessaires et définition de certaines constantes de couleur.


BLUE = (0,0,255)
BLACK = (0,0,0)
RED = (255,0,0)
YELLOW = (255,255,0)


#Définition du nombre de lignes et de colonnes du jeu Puissance 4.


ROW_COUNT = 6
COLUMN_COUNT = 7


#Définition d'une fonction created_board qui initialise un plateau de jeu vide avec des zéros.


def created_board():
    board = np.zeros((ROW_COUNT, COLUMN_COUNT))
    return board


#Fonction drop_piece pour déposer une pièce (jeton) sur le plateau.


def drop_piece(board, row, col, piece):
    board[row][col] = piece


#Fonction is_valid_location pour vérifier si une colonne est valide pour placer une pièce.
def is_valid_location(board, col):
    return board[ROW_COUNT-1][col] == 0


#Fonction get_next_open_row pour obtenir la première ligne ouverte dans une colonne donnée.


def get_next_open_row(board, col):
    for r in range(ROW_COUNT):
        if board[r][col] == 0:
            return r


#Fonction print_board pour afficher le plateau de jeu avec une orientation inversée (rotation de 180 degrés).
def print_board(board):
    print(np.flip(board, 0))


#Fonction winning_move pour vérifier si le joueur ayant la pièce spécifiée a gagné.
def winning_move(board, piece):
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT):
            if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] == piece:
                return True


# Fonction draw_board pour dessiner le plateau de jeu avec les pièces dans la fenêtre Pygame.
def draw_board(board):
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            pygame.draw.rect(screen, BLUE, (c*SQUARESIZE, r*SQUARESIZE+SQUARESIZE, SQUARESIZE, SQUARESIZE), width=0)
            pygame.draw.circle(screen, BLACK, (int(c*SQUARESIZE+SQUARESIZE/2), int(r*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)),
RADIUS)

    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            if board[r][c] == 1:
                pygame.draw.circle(screen, RED, (int(c*SQUARESIZE+SQUARESIZE/2), height-int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
            elif board[r][c] == 2:
                pygame.draw.circle(screen, YELLOW, (int(c*SQUARESIZE+SQUARESIZE/2), height-int(r*SQUARESIZE+SQUARESIZE/2)),
RADIUS)
```

```
    pygame.display.update()


#Initialisation du plateau, du statut du jeu (game_over), et du tour actuel.
board = created_board()
game_over = False
turn = 0


#Initialisation de Pygame.
pygame.init()


#Définition des dimensions de la fenêtre Pygame.
SQUARESIZE = 90
width = COLUMN_COUNT * SQUARESIZE
height = (ROW_COUNT+1) * SQUARESIZE


#Définition de la taille de la fenêtre Pygame, du rayon des pièces, création de la fenêtre, dessin du plateau et mise à jour de l'affichage.
size = (width, height)

RADIUS = int(SQUARESIZE/2 - 5)

screen = pygame.display.set_mode(size)

draw_board(board)

pygame.display.update()


#Définition d'une police pour le texte dans la fenêtre Pygame.


myfont = pygame.font.SysFont("monospace", 75)


#Boucle principale du jeu, qui gère les événements Pygame, tels que la fermeture de la fenêtre.


while not game_over:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()


#Gestion du mouvement de la souris pour afficher la position potentielle de la pièce à placer.


        if event.type == pygame.MOUSEMOTION:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            posx = event.pos[0]
            if turn == 0:
                pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE/2)), RADIUS)
            else:
                pygame.draw.circle(screen, YELLOW, (posx, int(SQUARESIZE/2)), RADIUS)
        pygame.display.update()


#Gestion du clic de souris pour placer une pièce lorsque c'est le tour du joueur.

python


        if event.type == pygame.MOUSEBUTTONDOWN:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            # print(event.pos)
            #Ask for palyer 1 input
            if turn == 0:
                posx = event.pos[0]
                col = int(math.floor(posx/SQUARESIZE))


#Vérification si le mouvement est valide, mise à jour du plateau, vérification de la victoire du joueur 1.


                if is_valid_location(board, col):
                    row = get_next_open_row(board, col)
                    drop_piece(board, row, col, 1)

                    if winning_move(board, 1):
                        label = myfont.render("Player 1 win!", 1, RED)
                        screen.blit(label, (40, 10))
                        game_over = True
```

```
# Tour du joueur 2, vérification de la validité du mouvement, mise à jour du plateau, vérification de la victoire du joueur 2.
        else:
            posx = event.pos[0]
            col = int(math.floor(posx/SQUARESIZE))

            if is_valid_location(board, col):
                row = get_next_open_row(board, col)
                drop_piece(board, row, col, 2)

                if winning_move(board, 2):
                    label = myfont.render("Player 2 win!", 1, YELLOW)
                    screen.blit(label, (40, 10))
                    game_over = True


#Mise à jour du plateau après chaque coup et changement de tour.


        draw_board(board)
        turn +=1
        turn = turn % 2


#Attente de 3 secondes après la fin du jeu avant de quitter


        if game_over:
            pygame.time.wait(3000)
```

Codes sources joueur vs IA basique:

```
import numpy as np
import pygame
import sys
import math
import random

#Import des bibliothèques nécessaires et définition de certaines constantes de couleur.

BLUE = (0,0,255)
BLACK = (0,0,0)
RED = (255,0,0)
YELLOW = (255,255,0)


#Définition du nombre de lignes et de colonnes du jeu Puissance 4.


ROW_COUNT = 6


COLUMN_COUNT = 7


#Définition de variables pour représenter le joueur humain (PLAYER), #l'intelligence artificielle (AI), les pièces du joueur et de l'IA, et des
#constantes pour la taille de la fenêtre de vérification de victoire.
PLAYER = 0
AI = 1

PLAYER_PIECE = 1
AI_PIECE = 2
EMPTY = 0

WINDOW_LENGTH = 4



#Définition d'une fonction created_board pour initialiser un plateau de jeu vide.
def created_board():
    board = np.zeros((ROW_COUNT, COLUMN_COUNT))
    return board
```

```
#Fonction drop_piece pour placer une pièce sur le plateau.

def drop_piece(board, row, col, piece):
    board[row][col] = piece


#Fonction is_valid_location pour vérifier si une colonne est valide pour placer #une pièce.
def is_valid_location(board, col):
    return board[ROW_COUNT-1][col] == 0


#Fonction get_next_open_row pour obtenir la première ligne ouverte dans une colonne donnée.

def get_next_open_row(board, col):
    for r in range(ROW_COUNT):
        if board[r][col] == 0:
            return r


#Fonction print_board pour afficher le plateau de jeu avec une orientation inversée (rotation de 180 degrés).
def print_board(board):
    print(np.flip(board, 0))



#Fonction winning_move pour vérifier si le joueur ayant la pièce spécifiée a gagné.
def winning_move(board, piece):
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT):
            if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] == piece:
                return True

    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3][c] == piece:
                return True

    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and board[r+3][c+3] == piece:
                return True

    for c in range(COLUMN_COUNT-3):
        for r in range(3, ROW_COUNT):
            if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and board[r-3][c+3] == piece:
                return True


#Fonction evaluate_window pour évaluer une fenêtre donnée et attribuer un score en fonction des pièces présentes.

def evaluate_window(window, piece):
    score = 0
    opp_piece = PLAYER_PIECE
    if piece == PLAYER_PIECE:
        opp_piece = AI_PIECE

    if window.count(piece) == 4:
        score += 100
    elif window.count(piece) == 3 and window.count(EMPTY) == 1:
        score += 10
    elif window.count(piece) == 2 and window.count(EMPTY) == 2:
        score += 10

    if window.count(opp_piece) == 3 and window.count(EMPTY) == 1:
        score -= 80

    return score


#Fonction score_position pour attribuer un score global à la position actuelle du plateau.


def score_position(board, piece):
    score = 0
    center_array = [int(i) for i in list(board[:, COLUMN_COUNT//2])]
    center_count = center_array.count(piece)
    score += center_count * 6
    for r in range(ROW_COUNT):
        row_array = [int(i) for i in list(board[r,:])]
```

```
            for c in range(COLUMN_COUNT-3):
                window = row_array[c:c+WINDOW_LENGTH]
                score += evaluate_window(window, piece)
        for c in range(COLUMN_COUNT):
            col_array = [int(i) for i in list(board[:,c])]
            for r in range(ROW_COUNT-3):
                window = col_array[r:r+WINDOW_LENGTH]
                score += evaluate_window(window, piece)


        for r in range(ROW_COUNT-3):
            for c in range(COLUMN_COUNT-3):
                window = [board[r+i][c+i] for i in range(WINDOW_LENGTH)]
                score += evaluate_window(window, piece)
        for r in range(ROW_COUNT-3):
            for c in range(COLUMN_COUNT-3):
                window = [board[r+3-i][c+i] for i in range(WINDOW_LENGTH)]
                score += evaluate_window(window, piece)

        return score


#Fonction get_valid_location pour obtenir toutes les colonnes valides où une pièce peut être placée.

def get_valid_location(board):
    valid_location = []
    for col in range(COLUMN_COUNT):
        if is_valid_location(board, col):
            valid_location.append(col)
    return valid_location


#Fonction pick_best_move pour choisir le meilleur coup possible pour l'IA en utilisant l'algorithme minimax simplifié.

def pick_best_move(board, piece):
    valid_location = get_valid_location(board)
    best_score = -1000
    best_col = random.choice(valid_location)
    for col in valid_location:
        row = get_next_open_row(board, col)
        temp_board = board.copy()
        drop_piece(temp_board, row, col, piece)
        score = score_position(temp_board, piece)
        if score > best_score:
            best_score = score
            best_col = col

    return best_col


# Fonction draw_board pour dessiner le plateau de jeu et les pièces en utilisant la bibliothèque Pygame.
def draw_board(board):
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            pygame.draw.rect(screen, BLUE, (c*SQUARESIZE, r*SQUARESIZE+SQUARESIZE, SQUARESIZE, SQUARESIZE), width=0)
            pygame.draw.circle(screen, BLACK, (int(c*SQUARESIZE+SQUARESIZE/2), int(r*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)),
RADIUS)

    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            if board[r][c] == PLAYER_PIECE:
                pygame.draw.circle(screen, RED, (int(c*SQUARESIZE+SQUARESIZE/2), height-int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
            elif board[r][c] == AI_PIECE:
                pygame.draw.circle(screen, YELLOW, (int(c*SQUARESIZE+SQUARESIZE/2), height-int(r*SQUARESIZE+SQUARESIZE/2)),
RADIUS)
    pygame.display.update()



# Initialisation des paramètres pour la fenêtre Pygame, y compris la taille, le plateau de jeu initial, le joueur qui commence aléatoirement.
board = created_board()
game_over = False

pygame.init()

SQUARESIZE = 90
width = COLUMN_COUNT * SQUARESIZE
height = (ROW_COUNT+1) * SQUARESIZE

size = (width, height)
```

```
RADIUS = int(SQUARESIZE/2 - 5)

screen = pygame.display.set_mode(size)
draw_board(board)
pygame.display.update()

myfont = pygame.font.SysFont("monospace", 75)

turn = random.randint(PLAYER, AI)


#Boucle principale du jeu qui gère les événements de la souris, les mouvements des joueurs, et l'affichage des résultats.

while not game_over:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

        if event.type == pygame.MOUSEMOTION:
            pygame.draw.rect(screen, BLACK, (0, 0, width, SQUARESIZE))
            posx = event.pos[0]
            if turn == PLAYER:
                pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE/2)), RADIUS)

        pygame.display.update()

        if event.type == pygame.MOUSEBUTTONDOWN:
            pygame.draw.rect(screen, BLACK, (0, 0, width, SQUARESIZE))
            if turn == PLAYER:
                posx = event.pos[0]
                col = int(math.floor(posx/SQUARESIZE))

                if is_valid_location(board, col):
                    row = get_next_open_row(board, col)
                    drop_piece(board, row, col, PLAYER_PIECE)

                    if winning_move(board, PLAYER_PIECE):
                        label = myfont.render("Player 1 win!", 1, RED)
                        screen.blit(label, (40, 10))
                        game_over = True

                    print_board(board)
                    draw_board(board)

                    turn += 1
                    turn = turn % 2

    if turn == AI and not game_over:
        col = pick_best_move(board, AI_PIECE)

        if is_valid_location(board, col):
            pygame.time.wait(1000)
            row = get_next_open_row(board, col)
            drop_piece(board, row, col, AI_PIECE)

            if winning_move(board, AI_PIECE):
                label = myfont.render("Player 2 win!", 1, YELLOW)
                screen.blit(label, (40, 10))
                game_over = True

            draw_board(board)

            turn += 1
            turn = turn % 2

    if game_over:
        pygame.time.wait(3000)
```

Codes sources du fichier joueur vs Minimax:

```
import numpy as np
import pygame
import sys
import math
import random

# Importation des bibliothèques nécessaires et initialisation des couleurs
BLUE = (0,0,255)
BLACK = (0,0,0)
RED = (255,0,0)
YELLOW = (255,255,0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)

# Définition du nombre de lignes et de colonnes du jeu
ROW_COUNT = 6
COLUMN_COUNT = 7

# Définition des variables pour représenter les joueurs et les pièces
PLAYER = 0
AI = 1
PLAYER_PIECE = 1
AI_PIECE = 2
EMPTY = 0
WINDOW_LENGTH = 4

# Fonction pour créer un plateau de jeu vide
def created_board():
    board = np.zeros((ROW_COUNT, COLUMN_COUNT))
    return board

# Fonction pour placer une pièce dans une colonne donnée
def drop_piece(board, row, col, piece):
    board[row][col] = piece

# Fonction pour vérifier si une colonne est valide pour placer une pièce
def is_valid_location(board, col):
    return board[ROW_COUNT-1][col] == 0

# Fonction pour obtenir la première ligne ouverte dans une colonne donnée
def get_next_open_row(board, col):
    for r in range(ROW_COUNT):
        if board[r][col] == 0:
            return r

# Fonction pour afficher le plateau de jeu avec une orientation inversée
def print_board(board):
    print(np.flip(board, 0))

# Fonction pour définir la difficulté du jeu
def get_difficulty():
    difficulty = None
    myfontDifficulty = pygame.font.SysFont("monospace", 28)
    while difficulty is None:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                pos = pygame.mouse.get_pos()
                x, y = pos
                # Vérifier la zone de clic pour définir la difficulté
                if 50 < x < 150 and 200 < y < 300:
                    difficulty = 1  # Très facile
                elif 200 < x < 300 and 200 < y < 300:
                    difficulty = 2  # Facile
```

```
            elif 350 < x < 450 and 200 < y < 300:
                difficulty = 3  # Normal
            elif 500 < x < 600 and 200 < y < 300:
                difficulty = 4  # Difficile

        screen.fill(BLACK)
        # Dessiner les rectangles pour les niveaux de difficulté
        pygame.draw.rect(screen, GREEN, (50, 200, 100, 100))  # Très facile (vert)
        pygame.draw.rect(screen, YELLOW, (200, 200, 100, 100))  # Facile (jaune)
        pygame.draw.rect(screen, BLUE, (350, 200, 100, 100))  # Moyen (bleu)
        pygame.draw.rect(screen, RED, (500, 200, 100, 100))  # Difficile (rouge)
        # Dessiner les messages de difficulté
        label = myfontDifficulty.render("Very Easy", 1, WHITE)
        screen.blit(label, (20, 120))
        label = myfontDifficulty.render("Easy", 1, WHITE)
        screen.blit(label, (220, 120))
        label = myfontDifficulty.render("Normal", 1, WHITE)
        screen.blit(label, (350, 120))
        label = myfontDifficulty.render("Difficult", 1, WHITE)
        screen.blit(label, (470, 120))
        pygame.display.update()
    return difficulty

# Fonction pour vérifier si un joueur a gagné
def winning_move(board, piece):
    # Vérifier l'emplacement horizontal
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT):
            if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] == piece:
                return True
    # Vérifier l'emplacement vertical
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3][c] == piece:
                return True
    # Vérifier l'emplacement des diagonales positives
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and board[r+3][c+3] == piece:
                return True
    # Vérifier l'emplacement des diagonales négatives
    for c in range(COLUMN_COUNT-3):
        for r in range(3, ROW_COUNT):
            if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and board[r-3][c+3] == piece:
                return True

# Fonction pour évaluer une fenêtre donnée et attribuer un score en fonction des pièces présentes
def evaluate_window(window, piece):
    score = 0
    opp_piece = PLAYER_PIECE
    if piece == PLAYER_PIECE:
        opp_piece = AI_PIECE
    if window.count(piece) == 3 and window.count(EMPTY) == 1:
        score += 5
    elif window.count(piece) == 2 and window.count(EMPTY) == 2:
        score += 2
    if window.count(opp_piece) == 3 and window.count(EMPTY) == 1:
        score -= 4
    return score

# Fonction pour attribuer un score global à la position actuelle du plateau
def score_position(board, piece):
    score = 0
    # Score colonne centrale
    center_array = [int(i) for i in list(board[:, COLUMN_COUNT//2])]
    center_count = center_array.count(piece)
    score += center_count * 3
    # Score horizontal
    for r in range(ROW_COUNT):
```

```
        row_array = [int(i) for i in list(board[r,:])]
        for c in range(COLUMN_COUNT-3):
            window = row_array[c:c+WINDOW_LENGTH]
            score += evaluate_window(window, piece)
    # Score vertical
    for c in range(COLUMN_COUNT):
        col_array = [int(i) for i in list(board[:,c])]
        for r in range(ROW_COUNT-3):
            window = col_array[r:r+WINDOW_LENGTH]
            score += evaluate_window(window, piece)
    # Score diagonale positive
    for r in range(ROW_COUNT-3):
        for c in range(COLUMN_COUNT-3):
            window = [board[r+i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)
    # Score diagonale négative
    for r in range(ROW_COUNT-3):
        for c in range(COLUMN_COUNT-3):
            window = [board[r+3-i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)
    return score

# Fonction pour déterminer si le jeu est terminé
def is_terminal_node(board):
    return winning_move(board, PLAYER_PIECE) or winning_move(board, AI_PIECE) or len(get_valid_location(board)) == 0

# Algorithme minimax
def minimax(board, depth, maximizingPlayer):
    valid_location = get_valid_location(board)
    is_terminal = is_terminal_node(board)
    if depth == 0 or is_terminal:
        if is_terminal:
            if winning_move(board, AI_PIECE):
                return (None, 1000000000000000)
            elif winning_move(board, PLAYER_PIECE):
                return (None, -100000000000000)
            else:
                return (None, 0)
        else:
            return (None, score_position(board, AI_PIECE))
    if maximizingPlayer:
        value = -math.inf
        column = random.choice(valid_location)
        for col in valid_location:
            row = get_next_open_row(board, col)
            b_copy = board.copy()
            drop_piece(b_copy, row, col, AI_PIECE)
            new_score = minimax(b_copy, depth-1, False)[1]
            if new_score > value:
                value = new_score
                column = col
        return column, value
    else:
        value = math.inf
        column = random.choice(valid_location)
        for col in valid_location:
            row = get_next_open_row(board, col)
            b_copy = board.copy()
            drop_piece(b_copy, row, col, PLAYER_PIECE)
            new_score = minimax(b_copy, depth-1, True)[1]
            if new_score < value:
                value = new_score
                column = col
        return column, value

# Fonction pour obtenir toutes les colonnes valides
def get_valid_location(board):
    valid_location = []
    for col in range(COLUMN_COUNT):
```

```
        if is_valid_location(board, col):
            valid_location.append(col)
    return valid_location

# Fonction pour dessiner le plateau de jeu en utilisant Pygame
def draw_board(board):
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            pygame.draw.rect(screen, BLUE, (c*SQUARESIZE, r*SQUARESIZE+SQUARESIZE, SQUARESIZE, SQUARESIZE), width=0)
            pygame.draw.circle(screen, BLACK, (int(c*SQUARESIZE+SQUARESIZE/2), int(r*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)),
RADIUS)
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            if board[r][c] == PLAYER_PIECE:
                pygame.draw.circle(screen, RED, (int(c*SQUARESIZE+SQUARESIZE/2), height-int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
            elif board[r][c] == AI_PIECE:
                pygame.draw.circle(screen, YELLOW, (int(c*SQUARESIZE+SQUARESIZE/2), height-int(r*SQUARESIZE+SQUARESIZE/2)),
RADIUS)
    pygame.display.update()

# Fonction pour réinitialiser le jeu avec un nouveau plateau
def reset_game():
    return np.array([[0] * COLUMN_COUNT for _ in range(ROW_COUNT)])

# Initialisation du plateau
board = created_board()
# Initialisation des conditions de la boucle de jeu
game_over = False
# Initialisation de Pygame
pygame.init()
# Initialisation des variables pour la taille de la fenêtre et le rayon des pièces
SQUARESIZE = 90
width = COLUMN_COUNT * SQUARESIZE
height = (ROW_COUNT+1) * SQUARESIZE
RADIUS = int(SQUARESIZE/2 - 5)
# Initialisation de la fenêtre Pygame
size = (width, height)
screen = pygame.display.set_mode(size)
# Initialisation de la difficulté et de la profondeur du minimax
difficulty = get_difficulty()
depth = difficulty
# Sélection aléatoire du premier joueur
turn = random.randint(PLAYER, AI)

# Boucle principale du jeu
while not game_over:
    # Gestion des événements Pygame
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        # Suivi de la position de la souris pour le joueur actuel
        if event.type == pygame.MOUSEMOTION:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            posx = event.pos[0]
            if turn == PLAYER:
                pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE/2)), RADIUS)
            pygame.display.update()
        # Détection du clic de souris
        if event.type == pygame.MOUSEBUTTONDOWN:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            # Tour du joueur humain
            if turn == 0:
                posx = event.pos[0]
                col = int(math.floor(posx/SQUARESIZE))
                if is_valid_location(board, col):
                    row = get_next_open_row(board, col)
                    drop_piece(board, row, col, PLAYER_PIECE)
                    if winning_move(board, PLAYER_PIECE):
                        label = myfontWin.render("Player 1 win!", 1, RED)
```

```
                screen.blit(label, (40, 10))
                game_over = True
            draw_board(board)
            turn +=1
            turn = turn % 2
    # Tour de l'IA
    if turn == AI and not game_over:
        col, minimax_score = minimax(board, depth, True)
        if is_valid_location(board, col):
            pygame.time.wait(1000)
            row = get_next_open_row(board, col)
            drop_piece(board, row, col, AI_PIECE)
            if winning_move(board, AI_PIECE):
                label = myfontWin.render("Player 2 win!", 1, YELLOW)
                screen.blit(label, (40, 10))
                game_over = True
            draw_board(board)
            turn +=1
            turn = turn % 2
    # Affichage du message de fin de jeu et gestion de la relance du jeu
    if game_over:
        draw_board(board)
        pygame.time.wait(3000)
        replay_text = myfontReset.render("Press Space to play or N to quit", 1, WHITE)
        screen.blit(replay_text, (10, 150))
        pygame.display.update()
        replay_decision = None
        while replay_decision is None:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    sys.exit()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_SPACE:
                        replay_decision = True
                    elif event.key == pygame.K_n:
                        replay_decision = False
        if replay_decision:
            # Initialisation d'un nouveau plateau
            board = reset_game()
            game_over = False
            difficulty = get_difficulty()
            depth = difficulty
            turn = random.randint(PLAYER, AI)
            draw_board(board)
        else:
            sys.exit()
```

Sources :

https://fr.wikipedia.org/wiki/Algorithme_minimax

https://fr.wikipedia.org/wiki/%C3%89lagage_alpha-b%C3%AAta

https://docs.python.org/fr/3/tutorial/

https://www.pygame.org/docs/

https://youtu.be/y7AKtWGOPAE

https://youtu.be/l-hh51ncgDI