import numpy as np

import random

import pygame

import sys

import math

BLUE = (0,0,255)

BLACK = (0,0,0)

RED = (255,0,0)

YELLOW = (255,255,0)

ROW\_COUNT = 6

COLUMN\_COUNT = 7

PLAYER = 0

AI = 1

EMPTY = 0

PLAYER\_PIECE = 1

AI\_PIECE = 2

WINDOW\_LENGTH = 4

def create\_board():

board = np.zeros((ROW\_COUNT,COLUMN\_COUNT))

return board

def drop\_piece(board, row, col, piece):

board[row][col] = piece

def is\_valid\_location(board, col):

return board[ROW\_COUNT-1][col] == 0

def get\_next\_open\_row(board, col):

for r in range(ROW\_COUNT):

if board[r][col] == 0:

return r

def print\_board(board):

print(np.flip(board, 0))

def winning\_move(board, piece):

# Check horizontal locations for win

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

# Check vertical locations for win

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

# Check positively sloped diaganols

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

# Check negatively sloped diaganols

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

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 += 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

def score\_position(board, piece):

score = 0

## Score center column

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 posiive sloped diagonal

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

def is\_terminal\_node(board):

return winning\_move(board, PLAYER\_PIECE) or winning\_move(board, AI\_PIECE) or len(get\_valid\_locations(board)) == 0

def minimax(board, depth, alpha, beta, maximizingPlayer):

valid\_locations = get\_valid\_locations(board)

is\_terminal = is\_terminal\_node(board)

if depth == 0 or is\_terminal:

if is\_terminal:

if winning\_move(board, AI\_PIECE):

return (None, 100000000000000)

elif winning\_move(board, PLAYER\_PIECE):

return (None, -10000000000000)

else: # Game is over, no more valid moves

return (None, 0)

else: # Depth is zero

return (None, score\_position(board, AI\_PIECE))

if maximizingPlayer:

value = -math.inf

column = random.choice(valid\_locations)

for col in valid\_locations:

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, alpha, beta, False)[1]

if new\_score > value:

value = new\_score

column = col

alpha = max(alpha, value)

if alpha >= beta:

break

return column, value

else: # Minimizing player

value = math.inf

column = random.choice(valid\_locations)

for col in valid\_locations:

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, alpha, beta, True)[1]

if new\_score < value:

value = new\_score

column = col

beta = min(beta, value)

if alpha >= beta:

break

return column, value

def get\_valid\_locations(board):

valid\_locations = []

for col in range(COLUMN\_COUNT):

if is\_valid\_location(board, col):

valid\_locations.append(col)

return valid\_locations

def pick\_best\_move(board, piece):

valid\_locations = get\_valid\_locations(board)

best\_score = -10000

best\_col = random.choice(valid\_locations)

for col in valid\_locations:

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

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))

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()

board = create\_board()

print\_board(board)

game\_over = False

pygame.init()

SQUARESIZE = 100

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)

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))

#print(event.pos)

# Ask for Player 1 Input

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 wins!!", 1, RED)

screen.blit(label, (40,10))

game\_over = True

turn += 1

turn = turn % 2

print\_board(board)

draw\_board(board)

# # Ask for Player 2 Input

if turn == AI and not game\_over:

#col = random.randint(0, COLUMN\_COUNT-1)

#col = pick\_best\_move(board, AI\_PIECE)

col, minimax\_score = minimax(board, 5, -math.inf, math.inf, True)

if is\_valid\_location(board, col):

#pygame.time.wait(500)

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 wins!!", 1, YELLOW)

screen.blit(label, (40,10))

game\_over = True

print\_board(board)

draw\_board(board)

turn += 1

turn = turn % 2

if game\_over:

pygame.time.wait(3000)