import pygame  
import sys  
import time  
  
# board.py ve ui.py'den import  
from board import (BOARD\_SIZE, TILE\_SIZE, BOARD\_OFFSET\_X, BOARD\_OFFSET\_Y,  
 SCOREBOARD\_WIDTH, WIDTH, HEIGHT, FPS,  
 WOOD\_COLOR, BLACK, WHITE, ERROR\_OVERLAY\_COLOR, Piece)  
from ui import Button  
# ai.py'den import  
from ai import (ZOBRIST, initialize\_zobrist, get\_all\_possible\_moves,  
 get\_all\_possible\_capture\_moves, make\_ai\_move)  
  
pygame.init()  
  
# Fontlar  
FONT\_SMALL = pygame.font.SysFont('Arial', 20)  
FONT\_MEDIUM = pygame.font.SysFont('Arial', 28)  
FONT\_LARGE = pygame.font.SysFont('Arial', 54)  
  
PLAYERS = {  
 'Player1': {'name': 'White', 'color': WHITE},  
 'Player2': {'name': 'Black', 'color': BLACK}  
}  
  
def quit\_game():  
 pygame.quit()  
 sys.exit()  
  
# Zobrist hashing'i başlat  
initialize\_zobrist()  
  
class Fianco:  
 def \_\_init\_\_(self):  
 self.window = pygame.display.set\_mode((WIDTH, HEIGHT))  
 pygame.display.set\_caption('FIANCO GAME')  
 self.clock = pygame.time.Clock()  
  
 self.state = 'menu' # Başlangıç: Menü  
 self.human\_player = None  
 self.ai\_player = None  
  
 self.create\_buttons()  
  
 self.pieces = []  
 self.selected\_piece = None  
 self.current\_player = 'Player1'  
 self.game\_over = False  
 self.error\_message = ''  
 self.show\_error = False  
 self.winner = None  
 self.winner\_name = ''  
 self.must\_continue\_capture = False  
  
 self.captured\_white = []  
 self.captured\_black = []  
  
 # 10 dakika (ms cinsinden)  
 self.player\_times = {'Player1': 600000, 'Player2': 600000}  
 self.last\_update\_time = pygame.time.get\_ticks()  
  
 # AI & Arama Değişkenleri  
 self.ttable = {}  
 self.prune\_count = 0  
 self.nodes\_searched = 0  
 self.total\_prunes = 0  
 self.max\_prune\_per\_move = 0  
 self.tt\_accesses = 0  
 self.stats\_printed = False  
  
 self.killer\_moves = {}  
 self.iterative\_time\_limit = 2000 # 2 saniye  
 self.max\_depth = 20  
  
 def create\_buttons(self):  
 import pygame  
 menu\_button\_width = 200  
 menu\_button\_height = 60  
 menu\_button\_font = pygame.font.SysFont('Arial', 24)  
 menu\_button\_bg = WHITE  
 menu\_button\_text = BLACK  
  
 self.menu\_play\_white\_button = Button(  
 x=WIDTH // 2 - menu\_button\_width - 20,  
 y=HEIGHT // 2,  
 width=menu\_button\_width,  
 height=menu\_button\_height,  
 text='Play as White',  
 font=menu\_button\_font,  
 bg\_color=menu\_button\_bg,  
 text\_color=menu\_button\_text,  
 action=lambda: self.select\_color('white')  
 )  
 self.menu\_play\_black\_button = Button(  
 x=WIDTH // 2 + 20,  
 y=HEIGHT // 2,  
 width=menu\_button\_width,  
 height=menu\_button\_height,  
 text='Play as Black',  
 font=menu\_button\_font,  
 bg\_color=menu\_button\_bg,  
 text\_color=menu\_button\_text,  
 action=lambda: self.select\_color('black')  
 )  
  
 base\_y = BOARD\_OFFSET\_Y + BOARD\_SIZE \* TILE\_SIZE + 70  
 button\_width = 100  
 button\_height = 35  
 button\_font = pygame.font.SysFont('Arial', 20)  
 button\_bg\_color = WHITE  
 button\_text\_color = BLACK  
  
 self.restart\_button = Button(  
 x=BOARD\_OFFSET\_X, y=base\_y, width=button\_width, height=button\_height,  
 text='Restart', font=button\_font, bg\_color=button\_bg\_color, text\_color=button\_text\_color,  
 action=self.reset\_game  
 )  
 self.in\_game\_quit\_button = Button(  
 x=BOARD\_OFFSET\_X + button\_width + 10, y=base\_y, width=button\_width, height=button\_height,  
 text='Quit', font=button\_font, bg\_color=button\_bg\_color, text\_color=button\_text\_color,  
 action=quit\_game  
 )  
  
 game\_over\_button\_width = 200  
 game\_over\_button\_height = 60  
 game\_over\_button\_font = pygame.font.SysFont('Arial', 28)  
 self.game\_over\_restart\_button = Button(  
 x=WIDTH // 2 - game\_over\_button\_width // 2, y=HEIGHT // 2 + 20,  
 width=game\_over\_button\_width, height=game\_over\_button\_height,  
 text='Restart Game', font=game\_over\_button\_font, bg\_color=button\_bg\_color, text\_color=button\_text\_color,  
 action=self.reset\_game  
 )  
 self.game\_over\_quit\_button = Button(  
 x=WIDTH // 2 - game\_over\_button\_width // 2, y=HEIGHT // 2 + 100,  
 width=game\_over\_button\_width, height=game\_over\_button\_height,  
 text='Quit Game', font=game\_over\_button\_font, bg\_color=button\_bg\_color, text\_color=button\_text\_color,  
 action=quit\_game  
 )  
  
 close\_button\_font = pygame.font.SysFont('Arial', 24)  
 close\_button\_text = close\_button\_font.render('X', True, WHITE)  
 close\_button\_image = pygame.Surface((30, 30), pygame.SRCALPHA)  
 close\_button\_image.blit(close\_button\_text, (5, 0))  
  
 close\_button\_rect = pygame.Rect(0, 0, 30, 30)  
 close\_button\_rect.x = (self.window.get\_width() - 50)  
 close\_button\_rect.y = 10  
 self.close\_button = Button(  
 x=close\_button\_rect.x, y=close\_button\_rect.y,  
 width=close\_button\_rect.width, height=close\_button\_rect.height,  
 text='', font=None, bg\_color=None, text\_color=None,  
 action=self.close\_error\_message, image=close\_button\_image  
 )  
  
 def select\_color(self, color):  
 if color == 'white':  
 self.human\_player = 'Player1'  
 self.ai\_player = 'Player2'  
 else:  
 self.human\_player = 'Player2'  
 self.ai\_player = 'Player1'  
  
 self.state = 'game'  
 self.reset\_game()  
  
 def close\_error\_message(self):  
 self.show\_error = False  
 self.error\_message = ''  
  
 def create\_initial\_pieces(self):  
 self.pieces = []  
 # Player1 (White) initial pieces  
 for col in range(BOARD\_SIZE):  
 self.pieces.append(Piece(0, col, PLAYERS['Player1']['color'], 'Player1'))  
 for i in range(1, 4):  
 self.pieces.append(Piece(i, i, PLAYERS['Player1']['color'], 'Player1'))  
 self.pieces.append(Piece(i, BOARD\_SIZE - 1 - i, PLAYERS['Player1']['color'], 'Player1'))  
  
 # Player2 (Black) initial pieces  
 for col in range(BOARD\_SIZE):  
 self.pieces.append(Piece(BOARD\_SIZE - 1, col, PLAYERS['Player2']['color'], 'Player2'))  
 for i in range(1, 4):  
 self.pieces.append(Piece(BOARD\_SIZE - 1 - i, i, PLAYERS['Player2']['color'], 'Player2'))  
 self.pieces.append(Piece(BOARD\_SIZE - 1 - i, BOARD\_SIZE - 1 - i, PLAYERS['Player2']['color'], 'Player2'))  
  
 def draw\_board(self):  
 self.window.fill(WOOD\_COLOR)  
 for row in range(BOARD\_SIZE + 1):  
 pygame.draw.line(  
 self.window, BLACK,  
 (BOARD\_OFFSET\_X, BOARD\_OFFSET\_Y + row \* TILE\_SIZE),  
 (BOARD\_OFFSET\_X + BOARD\_SIZE \* TILE\_SIZE, BOARD\_OFFSET\_Y + row \* TILE\_SIZE),  
 1  
 )  
 for col in range(BOARD\_SIZE + 1):  
 pygame.draw.line(  
 self.window, BLACK,  
 (BOARD\_OFFSET\_X + col \* TILE\_SIZE, BOARD\_OFFSET\_Y),  
 (BOARD\_OFFSET\_X + col \* TILE\_SIZE, BOARD\_OFFSET\_Y + BOARD\_SIZE \* TILE\_SIZE),  
 1  
 )  
  
 if self.selected\_piece:  
 x = self.selected\_piece.x  
 y = self.selected\_piece.y  
 radius = TILE\_SIZE // 2 - 5  
 pygame.draw.circle(self.window, (255, 215, 0), (x, y), radius + 3, 3)  
  
 for piece in self.pieces:  
 piece.draw(self.window)  
  
 self.draw\_captured\_pieces()  
  
 player\_info = PLAYERS[self.current\_player]  
 turn\_text = FONT\_MEDIUM.render(f"{player\_info['name']}'s Turn", True, player\_info['color'])  
 turn\_rect = turn\_text.get\_rect(  
 center=(BOARD\_OFFSET\_X + (BOARD\_SIZE \* TILE\_SIZE) // 2,  
 BOARD\_OFFSET\_Y + BOARD\_SIZE \* TILE\_SIZE + 30))  
 self.window.blit(turn\_text, turn\_rect)  
  
 def draw\_captured\_pieces(self):  
 # White captured  
 start\_x\_white = BOARD\_OFFSET\_X // 2  
 start\_y\_white = BOARD\_OFFSET\_Y  
 for index, piece in enumerate(self.captured\_white):  
 x = start\_x\_white  
 y = start\_y\_white + index \* (TILE\_SIZE // 2)  
 radius = TILE\_SIZE // 3  
 if y + radius > BOARD\_OFFSET\_Y + BOARD\_SIZE \* TILE\_SIZE:  
 break  
 pygame.draw.circle(self.window, PLAYERS['Player1']['color'], (x, y), radius)  
  
 # Black captured  
 start\_x\_black = BOARD\_OFFSET\_X + BOARD\_SIZE \* TILE\_SIZE + (BOARD\_OFFSET\_X // 2)  
 start\_y\_black = BOARD\_OFFSET\_Y  
 for index, piece in enumerate(self.captured\_black):  
 x = start\_x\_black  
 y = start\_y\_black + index \* (TILE\_SIZE // 2)  
 radius = TILE\_SIZE // 3  
 if y + radius > BOARD\_OFFSET\_Y + BOARD\_SIZE \* TILE\_SIZE:  
 break  
 pygame.draw.circle(self.window, PLAYERS['Player2']['color'], (x, y), radius)  
  
 def draw\_timers(self):  
 sidebar\_x = BOARD\_OFFSET\_X \* 2 + BOARD\_SIZE \* TILE\_SIZE + 10  
 top\_margin = 20  
 box\_width = SCOREBOARD\_WIDTH - 20  
 box\_height = 80  
 box\_spacing = 20  
  
 box\_y\_p1 = top\_margin  
 box\_y\_p2 = box\_y\_p1 + box\_height + box\_spacing  
  
 pygame.draw.rect(self.window, WHITE,  
 (BOARD\_OFFSET\_X \* 2 + BOARD\_SIZE \* TILE\_SIZE, 0, SCOREBOARD\_WIDTH, HEIGHT))  
  
 pygame.draw.rect(self.window, (230, 230, 230),  
 (sidebar\_x, box\_y\_p1, box\_width, box\_height))  
 pygame.draw.rect(self.window, BLACK,  
 (sidebar\_x, box\_y\_p1, box\_width, box\_height), 2)  
  
 pygame.draw.rect(self.window, (230, 230, 230),  
 (sidebar\_x, box\_y\_p2, box\_width, box\_height))  
 pygame.draw.rect(self.window, BLACK,  
 (sidebar\_x, box\_y\_p2, box\_width, box\_height), 2)  
  
 time\_left\_p1 = self.player\_times['Player1'] // 1000  
 m1 = time\_left\_p1 // 60  
 s1 = time\_left\_p1 % 60  
 time\_str\_p1 = f"{m1:02d}:{s1:02d}"  
  
 time\_left\_p2 = self.player\_times['Player2'] // 1000  
 m2 = time\_left\_p2 // 60  
 s2 = time\_left\_p2 % 60  
 time\_str\_p2 = f"{m2:02d}:{s2:02d}"  
  
 player\_name\_p1 = FONT\_MEDIUM.render("WHITE", True, BLACK)  
 player\_name\_p2 = FONT\_MEDIUM.render("BLACK", True, BLACK)  
 time\_font = pygame.font.SysFont('Arial', 32)  
  
 time\_surf\_p1 = time\_font.render(time\_str\_p1, True, BLACK)  
 time\_surf\_p2 = time\_font.render(time\_str\_p2, True, BLACK)  
  
 name\_rect\_p1 = player\_name\_p1.get\_rect(center=(sidebar\_x + box\_width // 2, box\_y\_p1 + 20))  
 self.window.blit(player\_name\_p1, name\_rect\_p1)  
 time\_rect\_p1 = time\_surf\_p1.get\_rect(center=(sidebar\_x + box\_width // 2, box\_y\_p1 + box\_height - 20))  
 self.window.blit(time\_surf\_p1, time\_rect\_p1)  
  
 name\_rect\_p2 = player\_name\_p2.get\_rect(center=(sidebar\_x + box\_width // 2, box\_y\_p2 + 20))  
 self.window.blit(player\_name\_p2, name\_rect\_p2)  
 time\_rect\_p2 = time\_surf\_p2.get\_rect(center=(sidebar\_x + box\_width // 2, box\_y\_p2 + box\_height - 20))  
 self.window.blit(time\_surf\_p2, time\_rect\_p2)  
  
 def display\_error\_message(self, message, close\_button):  
 overlay = pygame.Surface((WIDTH, HEIGHT), pygame.SRCALPHA)  
 overlay.fill(ERROR\_OVERLAY\_COLOR)  
 self.window.blit(overlay, (0, 0))  
  
 error\_text = FONT\_MEDIUM.render(message, True, WHITE)  
 error\_rect = error\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2))  
 self.window.blit(error\_text, error\_rect)  
  
 close\_button.draw(self.window)  
  
 def get\_piece\_at\_position(self, row, col):  
 for p in self.pieces:  
 if p.row == row and p.col == col:  
 return p  
 return None  
  
 def get\_mouse\_board\_position(self, pos):  
 x, y = pos  
 x -= BOARD\_OFFSET\_X  
 y -= BOARD\_OFFSET\_Y  
 c = x // TILE\_SIZE  
 r = y // TILE\_SIZE  
 return r, c  
  
 def remove\_piece(self, piece):  
 self.pieces.remove(piece)  
 if piece.owner == 'Player1':  
 self.captured\_white.append(piece)  
 else:  
 self.captured\_black.append(piece)  
 piece.row = None  
 piece.col = None  
 piece.reset\_position()  
  
 def is\_valid\_move(self, piece, r, c):  
 # aynen eski haliniz veya ufak iyileştirmeler  
 if piece.row == r and piece.col == c:  
 return False, "", None  
 if r < 0 or r >= BOARD\_SIZE or c < 0 or c >= BOARD\_SIZE:  
 return False, "", None  
 occ = self.get\_piece\_at\_position(r, c)  
 if occ:  
 return False, "", None  
  
 if piece.owner == 'Player1':  
 fw = 1  
 else:  
 fw = -1  
  
 dr = r - piece.row  
 dc = c - piece.col  
  
 # Geri gitme kontrolü (Fianco'da var mı yok mu? Mevcut kodda -fw engeli var).  
 if dr == -fw:  
 return False, "", None  
  
 # Normal yürüyüş  
 if dr == fw and dc == 0:  
 return True, "", None  
 # Yan hareket  
 elif dr == 0 and abs(dc) == 1:  
 return True, "", None  
 # Capture hamlesi  
 elif dr == fw \* 2 and abs(dc) == 2:  
 middle\_row = piece.row + fw  
 middle\_col = piece.col + (dc // 2)  
 mid\_piece = self.get\_piece\_at\_position(middle\_row, middle\_col)  
 if mid\_piece and mid\_piece.owner != piece.owner:  
 return True, "", mid\_piece  
  
 return False, "", None  
  
 def check\_for\_win(self):  
 for piece in self.pieces:  
 if piece.owner == 'Player1' and piece.row == BOARD\_SIZE - 1:  
 return 'Player1'  
 elif piece.owner == 'Player2' and piece.row == 0:  
 return 'Player2'  
 return None  
  
 def check\_for\_piece\_depletion(self):  
 p1 = [p for p in self.pieces if p.owner == 'Player1']  
 p2 = [p for p in self.pieces if p.owner == 'Player2']  
 if not p1:  
 return 'Player2'  
 elif not p2:  
 return 'Player1'  
 return None  
  
 def reset\_game(self):  
 self.create\_initial\_pieces()  
 self.selected\_piece = None  
 self.current\_player = self.human\_player  
 self.game\_over = False  
 self.error\_message = ''  
 self.show\_error = False  
 self.winner = None  
 self.winner\_name = ''  
 self.must\_continue\_capture = False  
 self.captured\_white.clear()  
 self.captured\_black.clear()  
 self.state = 'game'  
 self.player\_times['Player1'] = 600000  
 self.player\_times['Player2'] = 600000  
 self.last\_update\_time = pygame.time.get\_ticks()  
  
 # TT ve istatistikleri sıfırla  
 self.ttable.clear()  
 self.prune\_count = 0  
 self.nodes\_searched = 0  
 self.total\_prunes = 0  
 self.max\_prune\_per\_move = 0  
 self.tt\_accesses = 0  
 self.killer\_moves.clear()  
 self.stats\_printed = False  
  
 def make\_move(self, move, store\_previous\_state=False):  
 piece, r, c, captured = move  
 prev = None  
 if store\_previous\_state:  
 prev = {  
 'piece': piece,  
 'piece\_row': piece.row,  
 'piece\_col': piece.col,  
 'captured\_piece': captured,  
 'captured\_row': captured.row if captured else None,  
 'captured\_col': captured.col if captured else None,  
 'current\_player': self.current\_player,  
 'must\_continue\_capture': self.must\_continue\_capture,  
 'game\_over': self.game\_over,  
 'winner': self.winner,  
 'winner\_name': self.winner\_name,  
 'state': self.state,  
 'selected\_piece': self.selected\_piece  
 }  
  
 piece.row = r  
 piece.col = c  
 piece.reset\_position()  
  
 if captured and captured in self.pieces:  
 self.remove\_piece(captured)  
  
 w = self.check\_for\_win()  
 if not w:  
 w = self.check\_for\_piece\_depletion()  
 if w:  
 self.winner = w  
 self.game\_over = True  
 self.state = 'game\_over'  
 self.winner\_name = PLAYERS[w]['name']  
 else:  
 if captured and self.has\_available\_captures(piece):  
 self.must\_continue\_capture = True  
 self.selected\_piece = piece  
 else:  
 self.must\_continue\_capture = False  
 self.selected\_piece = None  
 if not self.must\_continue\_capture:  
 self.current\_player = 'Player2' if self.current\_player == 'Player1' else 'Player1'  
  
 if store\_previous\_state:  
 return prev  
  
 # Burada eğer oyun bitmediyse ve şu an AI sırası ise:  
 if not self.game\_over and self.current\_player == self.ai\_player:  
 make\_ai\_move(self)  
  
 def unmake\_move(self, prev):  
 piece = prev['piece']  
 piece.row = prev['piece\_row']  
 piece.col = prev['piece\_col']  
 piece.reset\_position()  
  
 captured\_piece = prev['captured\_piece']  
 if captured\_piece:  
 captured\_piece.row = prev['captured\_row']  
 captured\_piece.col = prev['captured\_col']  
 captured\_piece.reset\_position()  
 self.pieces.append(captured\_piece)  
 if captured\_piece.owner == 'Player1':  
 self.captured\_white.remove(captured\_piece)  
 else:  
 self.captured\_black.remove(captured\_piece)  
  
 self.current\_player = prev['current\_player']  
 self.must\_continue\_capture = prev['must\_continue\_capture']  
 self.game\_over = prev['game\_over']  
 self.winner = prev['winner']  
 self.winner\_name = prev['winner\_name']  
 self.state = prev['state']  
 self.selected\_piece = prev['selected\_piece']  
  
 def has\_available\_captures(self, piece):  
 # mevcutta sadece bir taş için capture var mı diye bakıyor  
 if piece.owner == 'Player1':  
 fw = 1  
 else:  
 fw = -1  
  
 for dc in [-2, 2]:  
 nr = piece.row + fw \* 2  
 nc = piece.col + dc  
 if 0 <= nr < BOARD\_SIZE and 0 <= nc < BOARD\_SIZE:  
 if not self.get\_piece\_at\_position(nr, nc):  
 mr = piece.row + fw  
 mc = piece.col + (dc // 2)  
 mid = self.get\_piece\_at\_position(mr, mc)  
 if mid and mid.owner != piece.owner:  
 return True  
 return False  
  
 def handle\_event\_manual(self, event):  
 if self.show\_error:  
 self.close\_button.handle\_event(event)  
  
 if self.state == 'menu':  
 self.menu\_play\_white\_button.handle\_event(event)  
 self.menu\_play\_black\_button.handle\_event(event)  
 return  
  
 if self.state == 'game':  
 self.in\_game\_quit\_button.handle\_event(event)  
 self.restart\_button.handle\_event(event)  
  
 if not self.game\_over:  
 if self.current\_player == self.human\_player:  
 if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:  
 pos = pygame.mouse.get\_pos()  
 if not self.selected\_piece:  
 # Henüz seçili taş yokken, tıklanan taş  
 for p in self.pieces:  
 rad = TILE\_SIZE // 2 - 5  
 dist = ((p.x - pos[0])\*\*2 + (p.y - pos[1])\*\*2)\*\*0.5  
 if dist <= rad and p.owner == self.current\_player:  
 self.selected\_piece = p  
 break  
 else:  
 # Seçili bir taş varsa, hedef kareyi anlamaya çalış  
 rr, cc = self.get\_mouse\_board\_position(pos)  
  
 # --- ZORUNLU YEME KONTROLÜ ---  
 # Şu anki oyuncu için capture hamleleri var mı?  
 from ai import get\_all\_possible\_capture\_moves  
 all\_capture\_moves = get\_all\_possible\_capture\_moves(self, self.current\_player)  
 # (ya da kendiniz get\_all\_possible\_capture\_moves yazıp fianco.py içine koyabilirsiniz)  
  
 if 0 <= rr < BOARD\_SIZE and 0 <= cc < BOARD\_SIZE:  
 clicked\_piece = self.get\_piece\_at\_position(rr, cc)  
 if clicked\_piece and clicked\_piece.owner == self.current\_player:  
 # Aynı oyuncunun başka taşını seçti  
 self.selected\_piece = clicked\_piece  
 else:  
 is\_ok, \_, cap = self.is\_valid\_move(self.selected\_piece, rr, cc)  
 if is\_ok:  
 # Eğer capture hamlesi varsa ve bu hamle capture değilse -> hata  
 if all\_capture\_moves and cap is None:  
 self.error\_message = 'Capture is mandatory!'  
 self.show\_error = True  
 self.error\_start\_time = pygame.time.get\_ticks()  
 else:  
 self.make\_move((self.selected\_piece, rr, cc, cap))  
 self.error\_message = ''  
 if self.game\_over:  
 return  
 else:  
 self.error\_message = 'Invalid Move!'  
 self.show\_error = True  
 self.error\_start\_time = pygame.time.get\_ticks()  
 else:  
 self.selected\_piece = None  
  
 elif self.state == 'game\_over':  
 self.game\_over\_restart\_button.handle\_event(event)  
 self.game\_over\_quit\_button.handle\_event(event)