Version 2

Requires:

```
pip install pygame
brew install stockfish
```

Code:

```
"""
Chess Game — Lichess-style UI  (v3 — all fixes applied)

Fixes:
 1. Move list: white & black on SAME ROW  (1. e4  e5)
 2. Piece rendering: platform-aware font fallback, letter-in-box if no Unicode glyphs
 3. Increment added to the player who JUST MOVED (not the opponent)
 4. Arrow keys navigate review in both directions
 5. History screen: confirm dialog before opening a saved-game review
"""

import pygame, sys, copy, random, time, json, os, platform, subprocess, threading, shutil
pygame.init()

# ─── Layout ──────────────────────────────────────────────────────────────────
BOARD_SIZE  = 640
PANEL_WIDTH = 300
WINDOW_W    = BOARD_SIZE + PANEL_WIDTH
WINDOW_H    = BOARD_SIZE
SQ          = BOARD_SIZE // 8

# ─── Colours ─────────────────────────────────────────────────────────────────
C_BG        = (22,  21,  18)
C_DARK_SQ   = (181, 136,  99)
C_LIGHT_SQ  = (240, 217, 181)
C_HIGHLIGHT = (205, 210, 106)
C_SELECTED  = (246, 246, 105)
C_PANEL     = (28,  27,  24)
C_PANEL2    = (38,  37,  33)
```

```
C_PANEL3    = (52,  50,  44)
C_TEXT      = (222, 220, 215)
C_TEXT2     = (140, 135, 125)
C_TEXT3     = (88,  85,  78)
C_ACCENT    = (128, 196, 127)
C_GOLD      = (255, 188,  66)
C_RED       = (210,  90,  90)
C_BLUE      = (100, 155, 220)
C_BTN       = (55,  53,  47)
C_BTN_H     = (75,  73,  65)
C_BTN_A     = (100, 155, 220)
C_CHECK     = (200,  55,  55)
C_BORDER    = (65,  62,  55)
C_SEP       = (50,  48,  43)

# ── Move classification colours ──────────────────────────────────────
CLF_BRILLIANT = (0,   200, 215)   # cyan
CLF_GREAT     = (90,  130, 170)   # blue-grey
CLF_BEST      = (100, 185, 100)   # green
CLF_GOOD      = (160, 210, 100)   # lime
CLF_MISTAKE   = (220, 140,  50)   # orange
CLF_BLUNDER   = (210,  70,  70)   # red
CLF_NONE      = (80,   78,  72)   # neutral

# name → (colour, symbol, description)
CLF_INFO = {
    'brilliant': (CLF_BRILLIANT, '!!', 'Brilliant — sacrifice with hidden value'),
    'great':     (CLF_GREAT,     '!',  'Great Move — only good reply'),
    'best':      (CLF_BEST,      '*',  'Best Move — top engine choice'),
    'good':      (CLF_GOOD,      'v',  'Good Move — solid play'),
    'mistake':   (CLF_MISTAKE,   '?',  'Mistake — clear positional loss'),
    'blunder':   (CLF_BLUNDER,   '??', 'Blunder — game-changing error'),
    'none':      (CLF_NONE,      '',   ''),
}

# ── Fonts ────────────────────────────────────────────────────────────
FNT_XL      = pygame.font.SysFont("segoeui",  32, bold=True)
FNT_LG      = pygame.font.SysFont("segoeui",  22, bold=True)
FNT_MD      = pygame.font.SysFont("segoeui",  17)
FNT_MDB     = pygame.font.SysFont("segoeui",  17, bold=True)
```

```
FNT_SM      = pygame.font.SysFont("segoeui",  14)
FNT_SMB     = pygame.font.SysFont("segoeui",  14, bold=True)
FNT_XS      = pygame.font.SysFont("segoeui",  12)
FNT_MONO    = pygame.font.SysFont("consolas", 14)
FNT_MONO_SM = pygame.font.SysFont("consolas", 13)
FNT_CLK     = pygame.font.SysFont("consolas", 28, bold=True)

# ── Piece rendering  (FIX #2) ──────────────────────────────────────────
UNICODE_SYMS = {'K':'♔','Q':'♕','R':'♖','B':'♗','N':'♘','P':'♙',
                'k':'♚','q':'♛','r':'♜','b':'♝','n':'♞','p':'♟'}

def _make_piece_font(size):
    plat = platform.system()
    if plat == "Windows":
        cands = ["segoeuisymbol","seguisym","segoeui","arial unicode ms"]
    elif plat == "Darwin":
        cands = ["apple symbols","arial unicode ms","lucida grande"]
    else:
        cands = ["dejavusans","symbola","freesans","unifont"]
    cands += [None]
    for name in cands:
        try:
            f = pygame.font.SysFont(name, size) if name else pygame.font.Font(None, size)
            surf = f.render("♔", True, (255,255,255))
            if surf.get_width() > 4:
                return f, True
        except Exception:
            pass
    return pygame.font.SysFont("consolas", size, bold=True), False

_PF_CACHE = {}
def _pfont(size):
    if size not in _PF_CACHE:
        _PF_CACHE[size] = _make_piece_font(size)
    return _PF_CACHE[size]

def draw_piece_at(surf, piece, px, py, size=SQ):
    is_white = piece.isupper()
    fs = int(size * 0.74)
    font, use_uni = _pfont(fs)
```

```
    if not use_uni:
        pad = max(4, size//8)
        bg  = (245,230,200) if is_white else (55,50,45)
        fg  = (40,35,30)    if is_white else (240,230,215)
        pygame.draw.rect(surf, bg,
                         (px+pad, py+pad, size-pad*2, size-pad*2),
                         border_radius=max(2, size//10))
        pygame.draw.rect(surf, (0,0,0),
                         (px+pad, py+pad, size-pad*2, size-pad*2),
                         1, border_radius=max(2, size//10))
        t = font.render(piece.upper(), True, fg)
        surf.blit(t, (px+size//2-t.get_width()//2, py+size//2-t.get_height()//2))
        return
    sym = UNICODE_SYMS.get(piece, '?')
    oc  = (230,228,224) if not is_white else (28,26,22)
    for ox,oy in ((-1,0),(1,0),(0,-1),(0,1)):
        o = font.render(sym, True, oc)
        surf.blit(o, (px+size//2-o.get_width()//2+ox, py+size//2-o.get_height()//2+oy))
    clr = (255,255,255) if is_white else (22,20,18)
    t   = font.render(sym, True, clr)
    surf.blit(t, (px+size//2-t.get_width()//2, py+size//2-t.get_height()//2))

# ── Helpers ──────────────────────────────────────────────────────────────
WHITE = 'w'; BLACK = 'b'
SAVE_FILE = os.path.join(os.path.dirname(os.path.abspath(__file__)), "chess_history.json")

TIME_CONTROLS = [
    {"label":"5 min",  "name":"Blitz",     "base":300,  "inc":0 },
    {"label":"10 min", "name":"Blitz",     "base":600,  "inc":0 },
    {"label":"15+10",  "name":"Rapid",     "base":900,  "inc":10},
    {"label":"30 min", "name":"Rapid",     "base":1800, "inc":0 },
    {"label":"60 min", "name":"Classical", "base":3600, "inc":0 },
    {"label":"90+30",  "name":"Classical", "base":5400, "inc":30},
]

PIECE_VALUES = {'P':100,'N':320,'B':330,'R':500,'Q':900,'K':20000}
PST = {
'P':[ 0, 0, 0, 0, 0, 0, 0, 0,50,50,50,50,50,50,50,50,
     10,10,20,30,30,20,10,10, 5, 5,10,25,25,10, 5, 5,
      0, 0, 0,20,20, 0, 0, 0, 5,-5,-10,0,0,-10,-5, 5,
```

```
      5,10,10,-20,-20,10,10,5, 0, 0, 0, 0, 0, 0, 0, 0],
'N':[-50,-40,-30,-30,-30,-30,-40,-50,-40,-20,0,0,0,0,-20,-40,
     -30,0,10,15,15,10,0,-30,-30,5,15,20,20,15,5,-30,
     -30,0,15,20,20,15,0,-30,-30,5,10,15,15,10,5,-30,
     -40,-20,0,5,5,0,-20,-40,-50,-40,-30,-30,-30,-30,-40,-50],
'B':[-20,-10,-10,-10,-10,-10,-10,-20,-10,0,0,0,0,0,0,-10,
     -10,0,5,10,10,5,0,-10,-10,5,5,10,10,5,5,-10,
     -10,0,10,10,10,10,0,-10,-10,10,10,10,10,10,10,-10,
     -10,5,0,0,0,0,5,-10,-20,-10,-10,-10,-10,-10,-10,-20],
'R':[ 0,0,0,0,0,0,0,0, 5,10,10,10,10,10,10,5,
     -5,0,0,0,0,0,0,-5,-5,0,0,0,0,0,0,-5,
     -5,0,0,0,0,0,0,-5,-5,0,0,0,0,0,0,-5,
     -5,0,0,0,0,0,0,-5, 0,0,0,5,5,0,0,0],
'Q':[-20,-10,-10,-5,-5,-10,-10,-20,-10,0,0,0,0,0,0,-10,
     -10,0,5,5,5,5,0,-10,-5,0,5,5,5,5,0,-5,
      0,0,5,5,5,5,0,-5,-10,5,5,5,5,5,0,-10,
     -10,0,5,0,0,0,0,-10,-20,-10,-10,-5,-5,-10,-10,-20],
'K':[-30,-40,-40,-50,-50,-40,-40,-30,-30,-40,-40,-50,-50,-40,-40,-30,
     -30,-40,-40,-50,-50,-40,-40,-30,-30,-40,-40,-50,-50,-40,-40,-30,
     -20,-30,-30,-40,-40,-30,-30,-20,-10,-20,-20,-20,-20,-20,-20,-10,
      20,20,0,0,0,0,20,20,20,30,10,0,0,10,30,20],
}

def rr(surf,color,rect,r=8,brd=0,bc=None):
    pygame.draw.rect(surf,color,rect,border_radius=r)
    if brd and bc: pygame.draw.rect(surf,bc,rect,brd,border_radius=r)

def tc(surf,txt,fnt,clr,cx,cy):
    t=fnt.render(str(txt),True,clr); surf.blit(t,(cx-t.get_width()//2,cy-t.get_height()//2))

def tl(surf,txt,fnt,clr,x,y):
    t=fnt.render(str(txt),True,clr); surf.blit(t,(x,y)); return t.get_width()

def fmt(secs):
    secs=max(0,int(secs)); m,s=divmod(secs,60); return f"{m}:{s:02d}"

# ═══════════════════════════════════════════════════════════════════════
# Chess Engine
# ═══════════════════════════════════════════════════════════════════════
class ChessBoard:
```

```
INIT = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
def __init__(self):
    self.board=[[None]*8 for _ in range(8)]; self.turn=WHITE
    self.castling={'K':True,'Q':True,'k':True,'q':True}
    self.ep_square=None; self.halfmove=0; self.fullmove=1
    self.history=[]; self.result=None; self.result_reason=''
    self._fen(self.INIT)

def _fen(self,fen):
    p=fen.split()
    for r,row in enumerate(p[0].split('/')):
        c=0
        for ch in row:
            if ch.isdigit(): c+=int(ch)
            else: self.board[r][c]=ch; c+=1
    self.turn=WHITE if p[1]=='w' else BLACK
    cs=p[2]; self.castling={'K':'K' in cs,'Q':'Q' in cs,'k':'k' in cs,'q':'q' in cs}
    if p[3]!='-': self.ep_square=(8-int(p[3][1]),ord(p[3][0])-ord('a'))
    self.halfmove=int(p[4]); self.fullmove=int(p[5])

def col(self,p): return WHITE if p and p.isupper() else (BLACK if p else None)
def pt(self,p):  return p.upper() if p else None

def pseudo(self,row,col):
    p=self.board[row][col]
    if not p: return []
    c=self.col(p); t=self.pt(p); mv=[]
    def ok(r,c2): return 0<=r<8 and 0<=c2<8
    def slide(dirs):
        for dr,dc in dirs:
            r,c2=row+dr,col+dc
            while ok(r,c2):
                tgt=self.board[r][c2]
                if tgt is None: mv.append((r,c2))
                elif self.col(tgt)!=c: mv.append((r,c2)); break
                else: break
                r+=dr; c2+=dc
    def jump(ds):
        for dr,dc in ds:
            r,c2=row+dr,col+dc
```

```
                if ok(r,c2) and self.col(self.board[r][c2])!=c: mv.append((r,c2))
        if   t=='R': slide([(1,0),(-1,0),(0,1),(0,-1)])
        elif t=='B': slide([(1,1),(1,-1),(-1,1),(-1,-1)])
        elif t=='Q': slide([(1,0),(-1,0),(0,1),(0,-1),(1,1),(1,-1),(-1,1),(-1,-1)])
        elif t=='N': jump([(2,1),(2,-1),(-2,1),(-2,-1),(1,2),(1,-2),(-1,2),(-1,-2)])
        elif t=='K':
            jump([(1,0),(-1,0),(0,1),(0,-1),(1,1),(1,-1),(-1,1),(-1,-1)])
            if c==WHITE and row==7 and col==4:
                if self.castling['K'] and not self.board[7][5] and not self.board[7][6]:
mv.append((7,6,'cK'))
                if self.castling['Q'] and not self.board[7][3] and not self.board[7][2] and
not self.board[7][1]: mv.append((7,2,'cQ'))
            elif c==BLACK and row==0 and col==4:
                if self.castling['k'] and not self.board[0][5] and not self.board[0][6]:
mv.append((0,6,'ck'))
                if self.castling['q'] and not self.board[0][3] and not self.board[0][2] and
not self.board[0][1]: mv.append((0,2,'cq'))
        elif t=='P':
            d=-1 if c==WHITE else 1; sr=6 if c==WHITE else 1; r2=row+d
            if ok(r2,col) and not self.board[r2][col]:
                mv.append((r2,col))
                if row==sr and not self.board[row+2*d][col]: mv.append((row+2*d,col))
            for dc in(-1,1):
                r2,c2=row+d,col+dc
                if ok(r2,c2):
                    tgt=self.board[r2][c2]
                    if tgt and self.col(tgt)!=c: mv.append((r2,c2))
                    elif self.ep_square==(r2,c2): mv.append((r2,c2,'ep'))
        return mv

    def _chk(self,color,board):
        king='K' if color==WHITE else 'k'
        kp=next(((r,c) for r in range(8) for c in range(8) if board[r][c]==king),None)
        if not kp: return True
        orig=self.board; self.board=board
        att=any((m[0],m[1])==kp for r in range(8) for c in range(8)
                if self.col(board[r][c]) is not None and self.col(board[r][c])!=color
                for m in self.pseudo(r,c))
        self.board=orig; return att
```

```
    def _apply(self,board,castling,ep,fr,fc,tr,tc,sp=None,promo='Q'):
        b=copy.deepcopy(board); p=b[fr][fc]; c=self.col(p); t2=p.upper() if p else None
        nc=dict(castling); ne=None
        if sp in('cK','cQ','ck','cq'):
            b[tr][tc]=b[fr][fc]; b[fr][fc]=None
            rm={'cK':(7,7,7,5),'cQ':(7,0,7,3),'ck':(0,7,0,5),'cq':(0,0,0,3)}
            rr2,rc,rt,rtc2=rm[sp]; b[rt][rtc2]=b[rr2][rc]; b[rr2][rc]=None
        elif sp=='ep':
            b[tr][tc]=b[fr][fc]; b[fr][fc]=None; b[fr][tc]=None
        else:
            b[tr][tc]=b[fr][fc]; b[fr][fc]=None
        if t2=='P' and (tr==0 or tr==7): b[tr][tc]=promo if c==WHITE else promo.lower()
        if p=='K': nc['K']=nc['Q']=False
        if p=='k': nc['k']=nc['q']=False
        for sq,key in[((7,0),'Q'),((7,7),'K'),((0,0),'q'),((0,7),'k')]:
            if (fr,fc)==sq or (tr,tc)==sq: nc[key]=False
        if t2=='P' and abs(tr-fr)==2: ne=((fr+tr)//2,fc)
        return b,nc,ne

    def legal(self,row,col):
        p=self.board[row][col]
        if not p: return []
        c=self.col(p); res=[]
        for m in self.pseudo(row,col):
            tr,tc2=m[0],m[1]; sp=m[2] if len(m)>2 else None
            if sp in('cK','cQ','ck','cq'):
                if self._chk(c,self.board): continue
                mc=5 if tc2==6 else 3
                b2,_,_=self._apply(self.board,self.castling,self.ep_square,row,col,row,mc)
                if self._chk(c,b2): continue
            nb,_,_=self._apply(self.board,self.castling,self.ep_square,row,col,tr,tc2,sp)
            if not self._chk(c,nb): res.append(m)
        return res

    def all_legal(self,color=None):
        if color is None: color=self.turn
        return [(r,c)+m for r in range(8) for c in range(8)
                if self.col(self.board[r][c])==color for m in self.legal(r,c)]

    def is_check(self):      return self._chk(self.turn,self.board)
```

```
def is_checkmate(self):  return not self.all_legal() and self.is_check()
def is_stalemate(self):  return not self.all_legal() and not self.is_check()
def is_insuf(self):
    ps=[(p.upper(),r,c) for r in range(8) for c in range(8)
        if (p:=self.board[r][c]) and p.upper()!='K']
    return len(ps)==0 or (len(ps)==1 and ps[0][0] in('N','B'))

def _san(self,fr,fc,tr,tc,sp,promo,board):
    p=board[fr][fc];
    if not p: return ''
    pt=p.upper(); CL='abcdefgh'; RL='87654321'; dest=CL[tc]+RL[tr]
    if sp in('cK','ck'): return 'O-O'
    if sp in('cQ','cq'): return 'O-O-O'
    cap='x' if board[tr][tc] or sp=='ep' else ''
    if pt=='P':
        s=(CL[fc]+cap+dest) if cap else dest
        if tr==0 or tr==7: s+='='+promo
        return s
    return pt+cap+dest

def fen(self):
    rows=[]
    for r in range(8):
        e=0; s2=''
        for c in range(8):
            p=self.board[r][c]
            if not p: e+=1
            else:
                if e: s2+=str(e); e=0
                s2+=p
        if e: s2+=str(e)
        rows.append(s2)
    f='/'.join(rows)+' '+('w' if self.turn==WHITE else 'b')+' '
    cs=''.join(k for k in('K','Q','k','q') if self.castling[k])
    f+=(cs or '-')+' '
    if self.ep_square: f+='abcdefgh'[self.ep_square[1]]+str(8-self.ep_square[0])
    else: f+='-'
    f+=f' {self.halfmove} {self.fullmove}'
    return f
```

```
    def make_move(self,fr,fc,tr,tc,sp=None,promo='Q'):
        p=self.board[fr][fc]; cap=self.board[tr][tc]
        if sp=='ep': cap=self.board[fr][tc]
        old_b=copy.deepcopy(self.board); san=self._san(fr,fc,tr,tc,sp,promo,old_b)
        nb,nc,ne=self._apply(self.board,self.castling,self.ep_square,fr,fc,tr,tc,sp,promo)

info={'from':(fr,fc),'to':(tr,tc),'piece':p,'captured':cap,'special':sp,'promo':promo,'san':sa
n}
        self.history.append({'move':info,'board':old_b,'castling':dict(self.castling),
                             'ep':self.ep_square,'hm':self.halfmove,'turn':self.turn})
        self.board=nb; self.castling=nc; self.ep_square=ne
        pt=p.upper() if p else None
        self.halfmove=0 if (pt=='P' or cap) else self.halfmove+1
        if self.turn==BLACK: self.fullmove+=1
        self.turn=BLACK if self.turn==WHITE else WHITE
        if   self.is_checkmate():  self.result=WHITE if self.turn==BLACK else BLACK;
self.result_reason='checkmate'
        elif self.is_stalemate():  self.result='draw'; self.result_reason='stalemate'
        elif self.is_insuf():      self.result='draw'; self.result_reason='insufficient
material'
        elif self.halfmove>=100:   self.result='draw'; self.result_reason='50-move rule'
        return info

# ═══════════════════════════════════════════════════════════════════════════
# Fallback Python AI  (used when Stockfish is unavailable)
# ═══════════════════════════════════════════════════════════════════════════
class AI:
    D={1:1,2:1,3:2,4:2,5:3,6:3,7:4,8:4,9:5,10:6,11:6}
    N={1:350,2:250,3:180,4:120,5:70,6:40,7:20,8:8,9:2,10:0,11:0}
    def __init__(self,lv=5): self.lv=lv
    def eval(self,cb):
        s=0
        for r in range(8):
            for c in range(8):
                p=cb.board[r][c]
                if not p: continue
                pt=p.upper(); v=PIECE_VALUES.get(pt,0); idx=r*8+c if p.isupper() else (7-
r)*8+c
                s+=(v+PST[pt][idx]) if p.isupper() else -(v+PST[pt][idx])
        # mobility removed for speed
```

```
        return s
    def _ord(self,cb,moves):
        def sc(m):
            fr,fc,tr,tc=m[0],m[1],m[2],m[3]; sp=m[4] if len(m)>4 else None
            cap=cb.board[tr][tc] if sp!='ep' else cb.board[m[0]][tc]
            return PIECE_VALUES.get((cap or'').upper(),0)-
PIECE_VALUES.get(cb.board[fr][fc].upper(),0)//10
        return sorted(moves,key=sc,reverse=True)
    def _cl(self,cb):
        n=ChessBoard.__new__(ChessBoard)
        n.board=copy.deepcopy(cb.board); n.turn=cb.turn; n.castling=dict(cb.castling)
        n.ep_square=cb.ep_square; n.halfmove=cb.halfmove; n.fullmove=cb.fullmove
        n.history=[]; n.result=cb.result; n.result_reason=''; return n
    def _ab(self,cb,d,a,b,mx):
        if d==0 or cb.result: return self.eval(cb)
        ms=cb.all_legal()
        if not ms: return (-99999 if mx else 99999) if cb.is_check() else 0
        ms=self._ord(cb,ms)
        if mx:
            v=-999999
            for m in ms:
                c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
                v=max(v,self._ab(c2,d-1,a,b,False)); a=max(a,v)
                if b<=a: break
            return v
        else:
            v=999999
            for m in ms:
                c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
                v=min(v,self._ab(c2,d-1,a,b,True)); b=min(b,v)
                if b<=a: break
            return v
    def best(self,cb):
        depth=self.D.get(self.lv,3); noise=self.N.get(self.lv,0)
        ms=cb.all_legal()
        if not ms: return None
        ms=self._ord(cb,ms); mx=(cb.turn==WHITE); bv=-999999 if mx else 999999; bms=[]
        for m in ms:
            c2=self._cl(cb); c2.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
            v=self._ab(c2,depth-1,-999999,999999,not mx)+random.randint(-noise,noise)
```

```
            if(mx and v>bv)or(not mx and v<bv): bv=v; bms=[m]
            elif v==bv: bms.append(m)
        return random.choice(bms) if bms else random.choice(ms)

# ═══════════════════════════════════════════════════════════════════════
# Stockfish AI  (async, non-blocking)
# ═══════════════════════════════════════════════════════════════════════
class StockfishAI:
    # Level → UCI_Elo
    LEVEL_ELO  = {1:500, 2:800, 3:1200, 4:1600, 5:1800,
                  6:2000, 7:2200, 8:2300, 9:2400, 10:2500, 11:2700}
    # Level → movetime (ms)
    LEVEL_TIME = {1:80, 2:100, 3:150, 4:200, 5:300,
                  6:500, 7:800, 8:1200, 9:1800, 10:2500, 11:4000}

    def __init__(self, level=5):
        self.level   = level
        self._proc   = None
        self._thread = None
        self._result = None   # (fr,fc,tr,tc,sp,promo) when ready
        self._lock   = threading.Lock()
        self.ready   = False
        self._init()

    # ── process management ──────────────────────────────────────────
    def _find(self):
        # 1) same folder as this script
        base = os.path.dirname(os.path.abspath(__file__))
        for name in ("stockfish","stockfish.exe",
                     "stockfish-windows-x86-64-avx2.exe",
                     "stockfish-windows-x86-64-modern.exe"):
            p = os.path.join(base, name)
            if os.path.isfile(p): return p
        # 2) system PATH  (covers brew install on macOS)
        return shutil.which("stockfish")

    def _init(self):
        path = self._find()
        if not path:
            print("[Stockfish] Not found — using fallback Python AI.")
```

```
            return
        try:
            self._proc = subprocess.Popen(
                [path],
                stdin=subprocess.PIPE, stdout=subprocess.PIPE,
                stderr=subprocess.DEVNULL, text=True, bufsize=1)
            self._send("uci");        self._wait("uciok")
            self._apply_level(self.level)
            self._send("isready");    self._wait("readyok")
            self.ready = True
            print(f"[Stockfish] Ready  level={self.level}  "
                  f"elo={self.LEVEL_ELO[self.level]}")
        except Exception as e:
            print(f"[Stockfish] Failed to start: {e}")
            self.ready = False

    def _send(self, cmd):
        if self._proc:
            self._proc.stdin.write(cmd + "\n")
            self._proc.stdin.flush()

    def _wait(self, token, timeout=10.0):
        deadline = time.time() + timeout
        while time.time() < deadline:
            line = self._proc.stdout.readline()
            if token in line: return line
        return ""

    def _apply_level(self, level):
        elo = self.LEVEL_ELO.get(level, 1750)
        self._send("setoption name UCI_LimitStrength value true")
        self._send(f"setoption name UCI_Elo value {elo}")

    def set_level(self, level):
        self.level = level
        if self.ready:
            self._apply_level(level)
            self._send("isready"); self._wait("readyok")

    def close(self):
```

```
    if self._proc:
        try: self._send("quit"); self._proc.terminate()
        except: pass
        self._proc = None
    self.ready = False

# ── async move request ──────────────────────────────────────────
def start_thinking(self, fen):
    if not self.ready: return
    with self._lock: self._result = None
    self._thread = threading.Thread(target=self._think, args=(fen,), daemon=True)
    self._thread.start()

def _think(self, fen):
    mt = self.LEVEL_TIME.get(self.level, 500)
    self._send(f"position fen {fen}")
    self._send(f"go movetime {mt}")
    line = self._wait("bestmove", timeout=mt/1000 + 8)
    parts = line.strip().split()
    if len(parts) >= 2 and parts[0] == "bestmove" and parts[1] != "(none)":
        with self._lock:
            self._result = self._parse(parts[1])

def is_thinking(self):
    return self._thread is not None and self._thread.is_alive()

def get_result(self):
    """Returns move tuple or None if still thinking."""
    if self.is_thinking(): return None
    with self._lock:
        r = self._result; self._result = None
    return r

# ── UCI string → internal move tuple ────────────────────────────
def _parse(self, uci):
    """
    'e2e4'  → (6,4,4,4, None, 'Q')
    'e7e8q' → (1,4,0,4, None, 'Q')   promotion
    'e1g1'  → (7,4,7,6, 'cK', 'Q')   castling
    """
```

```
        fc  = ord(uci[0]) - ord('a')
        fr  = 8 - int(uci[1])
        tc2 = ord(uci[2]) - ord('a')
        tr  = 8 - int(uci[3])
        promo = uci[4].upper() if len(uci) == 5 else 'Q'
        # Castling: king moves exactly 2 squares horizontally
        sp = None
        if   uci == 'e1g1': sp = 'cK'
        elif uci == 'e1c1': sp = 'cQ'
        elif uci == 'e8g8': sp = 'ck'
        elif uci == 'e8c8': sp = 'cq'
        return (fr, fc, tr, tc2, sp, promo)

# ═══════════════════════════════════════════════════════════════════════
# Review Analyser  — uses Stockfish to score every position in a game
# ═══════════════════════════════════════════════════════════════════════
class ReviewAnalyser:
    """
    Runs Stockfish analysis on every position in a game (in a background thread)
    and classifies each half-move as brilliant / great / best / good / mistake / blunder.

    Results are stored as:
      self.evals  : list[float]  cp score (centipawns, white-positive) AFTER move i
                    index 0 = initial position, index N = after move N
      self.classif: list[str]   classification for move i (1-based, index 0 unused)
      self.done   : bool
    """
    DEPTH      = 16          # analysis depth
    ANALYSIS_MOVETIME = 300  # ms per position

    def __init__(self, sf_path):
        self.sf_path  = sf_path
        self.evals    = []
        self.classif  = []
        self.done     = False
        self._thread  = None

    def analyse(self, rev_hist, rev_boards):
        """Start background analysis. rev_hist / rev_boards from _build_rev."""
        self.evals   = [None] * (len(rev_hist) + 1)
```

```
        self.classif = ['none'] * (len(rev_hist) + 1)
        self.done    = False
        self._thread = threading.Thread(
            target=self._run, args=(rev_hist, rev_boards), daemon=True)
        self._thread.start()

    def _score_fen(self, proc, fen):
        """Ask Stockfish for a centipawn score for this FEN. Returns float (white POV)."""
        proc.stdin.write(f"position fen {fen}\n")
        proc.stdin.write(f"go movetime {self.ANALYSIS_MOVETIME}\n")
        proc.stdin.flush()
        score = None
        mate  = None
        deadline = time.time() + self.ANALYSIS_MOVETIME/1000 + 5
        while time.time() < deadline:
            line = proc.stdout.readline().strip()
            if line.startswith("info") and "score" in line:
                parts = line.split()
                try:
                    si = parts.index("score")
                    kind = parts[si+1]
                    val  = int(parts[si+2])
                    if kind == "cp":   score = val
                    elif kind == "mate": mate = val
                except (ValueError, IndexError):
                    pass
            if line.startswith("bestmove"):
                break
        if mate is not None:
            return 30000 if mate > 0 else -30000
        return float(score) if score is not None else 0.0

    def _run(self, rev_hist, rev_boards):
        path = self.sf_path
        if not path or not os.path.isfile(path):
            path = shutil.which("stockfish")
        if not path:
            self.done = True; return
        try:
            proc = subprocess.Popen(
```

```
            [path],
            stdin=subprocess.PIPE, stdout=subprocess.PIPE,
            stderr=subprocess.DEVNULL, text=True, bufsize=1)
        proc.stdin.write("uci\n"); proc.stdin.flush()
        # wait for uciok
        deadline = time.time()+10
        while time.time()<deadline:
            if "uciok" in proc.stdout.readline(): break
        proc.stdin.write("setoption name UCI_LimitStrength value false\n")
        proc.stdin.write("isready\n"); proc.stdin.flush()
        deadline = time.time()+10
        while time.time()<deadline:
            if "readyok" in proc.stdout.readline(): break

        # Build a temporary ChessBoard to replay
        tmp = ChessBoard()
        self.evals[0] = self._score_fen(proc, tmp.fen())

        for i, h in enumerate(rev_hist):
            m = h['move']
            tmp.make_move(m['from'][0],m['from'][1],
                          m['to'][0],  m['to'][1],
                          m['special'], m['promo'])
            score = self._score_fen(proc, tmp.fen())
            self.evals[i+1] = score

        proc.stdin.write("quit\n"); proc.stdin.flush()
        proc.terminate()
    except Exception as e:
        print(f"[ReviewAnalyser] error: {e}")
    finally:
        self._classify(rev_hist)
        self.done = True

def _classify(self, rev_hist):
    """Classify each move given the eval before and after."""
    for i, h in enumerate(rev_hist):
        before = self.evals[i]
        after  = self.evals[i+1]
        if before is None or after is None:
```

```
            self.classif[i+1] = 'none'; continue
        color = h['move'].get('piece','?')
        # delta from the mover's perspective (positive = good for mover)
        if color and color.isupper():   # white moved
            delta = after - before   # higher after = better for white
        else:                           # black moved
            delta = before - after   # lower after = better for black (cp is white-POV)

        # Was the best move actually played?
        # We approximate: if delta >= -10 it's essentially best
        if delta >= -10:
            # Check for brilliant: mover sacrificed material yet eval improved
            cap = h['move'].get('captured')
            cap_val = {'P':100,'N':320,'B':330,'R':500,'Q':900}.get(
                (cap or '').upper(), 0)
            if cap_val == 0 and delta >= 50:
                self.classif[i+1] = 'brilliant'
            elif cap_val > 0 and delta >= 0:
                # sacrificed but still good → brilliant candidate
                self.classif[i+1] = 'brilliant'
            else:
                self.classif[i+1] = 'best'
        elif delta >= -30:
            self.classif[i+1] = 'great'
        elif delta >= -80:
            self.classif[i+1] = 'good'
        elif delta >= -200:
            self.classif[i+1] = 'mistake'
        else:
            self.classif[i+1] = 'blunder'

def get_eval(self, idx):
    """Return eval at position idx (0=start), or None if not ready yet."""
    if idx < len(self.evals): return self.evals[idx]
    return None

def get_classif(self, move_idx):
    """Return classification string for half-move move_idx (1-based)."""
    if move_idx < len(self.classif): return self.classif[move_idx]
    return 'none'
```

```
def load_hist():
    try:
        with open(SAVE_FILE,'r') as f: return json.load(f)
    except: return []

def save_hist(recs):
    try:
        with open(SAVE_FILE,'w') as f: json.dump(recs[-10:],f,indent=2)
    except: pass

def record_game(cb,mode,ai_lv,tc_label,wt,bt):
    recs=load_hist()
    rs="White wins" if cb.result==WHITE else ("Black wins" if cb.result==BLACK else "Draw")
    recs.append({'date':time.strftime('%Y-%m-%d %H:%M'),'mode':mode,'ai_level':ai_lv,
                 'tc':tc_label,'result':rs,'reason':cb.result_reason,
                 'moves':[h['move']['san'] for h in cb.history],
                 'white_time_left':round(wt,1),'black_time_left':round(bt,1),
                 'total_moves':len(cb.history)})
    save_hist(recs)

# ═══════════════════════════════════════════════════════════════════════
# Screen IDs
# ═══════════════════════════════════════════════════════════════════════
class S:
    MAIN=0; SETUP_BOT=1; SETUP_LOC=2; PLAYING=3; PROMOTION=4
    REVIEW=5; HISTORY=6; HIST_CONFIRM=7

# ═══════════════════════════════════════════════════════════════════════
# Game
# ═══════════════════════════════════════════════════════════════════════
class Game:
    def __init__(self):
        self.surf=pygame.display.set_mode((WINDOW_W,WINDOW_H))
        pygame.display.set_caption("Chess")
        self.clk=pygame.time.Clock()
        self.state=S.MAIN
        # setup
        self.opt_ai=5; self.opt_tc=0; self.opt_col=WHITE
        # game
        self.cb=None; self.ai=StockfishAI(level=5); self.mode=None
```

```
        self.player_col=WHITE; self.flipped=False
        # clocks
        self.wt=0.0; self.bt=0.0; self.inc=0
        self.clk_last=0.0; self.clk_run=False
        # board UI
        self.sel=None; self.ltgts=[]; self.lm=None
        self.drag_p=None; self.drag_pos=None; self.drag_fr=None
        # promo
        self.promo=None
        # ai
        self.ai_busy=False
        # review
        self.rev_hist=[]; self.rev_idx=0; self.rev_boards=[]
        self.rev_src=S.PLAYING  # where Back goes
        self.rev_analyser=None  # ReviewAnalyser instance
        # history
        self.hist_recs=[]; self.hist_scroll=0; self.hist_confirm=None
        # notif
        self.notif=''; self.notif_exp=0

    # ── coords ──────────────────────────────────────────────────────────
    def s2p(self,r,c):
        return ((7-c)*SQ,(7-r)*SQ) if self.flipped else (c*SQ,r*SQ)
    def p2s(self,x,y):
        if not(0<=x<BOARD_SIZE and 0<=y<BOARD_SIZE): return None,None
        return (7-y//SQ,7-x//SQ) if self.flipped else (y//SQ,x//SQ)

    # ── clock ───────────────────────────────────────────────────────────
    def tick(self):
        if not self.clk_run or not self.cb or self.cb.result: return
        now=time.time(); dt=now-self.clk_last; self.clk_last=now
        if self.cb.turn==WHITE: self.wt=max(0,self.wt-dt)
        else:                   self.bt=max(0,self.bt-dt)
        if self.wt<=0 or self.bt<=0:
            self.cb.result=BLACK if self.wt<=0 else WHITE
            self.cb.result_reason='timeout'
            self.clk_run=False; self._end()

    # FIX #3 — increment goes to the MOVER
    def _add_inc(self,mover):
```

```
        if self.inc<=0: return
        if mover==WHITE: self.wt+=self.inc
        else:            self.bt+=self.inc

    # ── lifecycle ──────────────────────────────────────────────────────────
    def start(self,mode):
        self.mode=mode; tc=TIME_CONTROLS[self.opt_tc]
        self.wt=float(tc['base']); self.bt=float(tc['base']); self.inc=tc['inc']
        self.cb=ChessBoard()
        # Close previous Stockfish process before creating a new one
        if isinstance(self.ai, StockfishAI): self.ai.close()
        self.ai=StockfishAI(level=self.opt_ai)
        # If Stockfish unavailable, fall back to Python AI
        if not self.ai.ready: self.ai=AI(self.opt_ai)
        self.sel=None; self.ltgts=[]; self.lm=None
        self.drag_p=None; self.promo=None; self.ai_busy=False
        self.clk_run=False; self.clk_last=time.time()
        if mode=='bot':
            self.player_col=self.opt_col; self.flipped=(self.opt_col==BLACK)
        else:
            self.player_col=WHITE; self.flipped=False
        self.state=S.PLAYING

    def _end(self):
        record_game(self.cb,self.mode,self.opt_ai,
                    TIME_CONTROLS[self.opt_tc]['label'],self.wt,self.bt)

    def open_review(self,history,src=S.PLAYING):
        self.rev_hist=history; self.rev_src=src
        self._build_rev()
        self.rev_idx=len(self.rev_hist); self.state=S.REVIEW
        # Start Stockfish analysis in background
        sf_path = shutil.which("stockfish")
        self.rev_analyser = ReviewAnalyser(sf_path)
        self.rev_analyser.analyse(self.rev_hist, self.rev_boards)

    def open_review_game(self):
        if self.cb and self.cb.history: self.open_review(self.cb.history[:],S.PLAYING)

    def _build_rev(self):
```

```
        self.rev_boards=[]
        tmp=ChessBoard(); self.rev_boards.append(copy.deepcopy(tmp.board))
        for h in self.rev_hist:
            m=h['move']

tmp.make_move(m['from'][0],m['from'][1],m['to'][0],m['to'][1],m['special'],m['promo'])
            self.rev_boards.append(copy.deepcopy(tmp.board))

    def open_hist(self):
        self.hist_recs=load_hist(); self.hist_scroll=0; self.hist_confirm=None
        self.state=S.HISTORY

    # ── move execution ──────────────────────────────────────────────
    def try_move(self,fr,fc,tr,tc2):
        legal=self.cb.legal(fr,fc)
        m=next((x for x in legal if x[0]==tr and x[1]==tc2),None)
        if m is None:
            p=self.cb.board[tr][tc2]
            if p and self.cb.col(p)==self.cb.turn:
                self.sel=(tr,tc2); self.ltgts=self.cb.legal(tr,tc2)
            else:
                self.sel=None; self.ltgts=[]
            return
        sp=m[2] if len(m)>2 else None
        if self.cb.pt(self.cb.board[fr][fc])=='P' and (tr==0 or tr==7):
            self.promo=(fr,fc,tr,tc2,sp); self.state=S.PROMOTION
            self.sel=None; self.ltgts=[]; return
        self._do(fr,fc,tr,tc2,sp,'Q')

    def _do(self,fr,fc,tr,tc2,sp,promo):
        mover=self.cb.turn
        info=self.cb.make_move(fr,fc,tr,tc2,sp,promo)
        self._add_inc(mover)                    # FIX #3
        self.lm=((fr,fc),(tr,tc2))
        self.sel=None; self.ltgts=[]
        self.clk_last=time.time()
        if not self.clk_run and len(self.cb.history)>=1: self.clk_run=True
        if self.cb.result: self.clk_run=False; self._end()
        self.notif=info['san']; self.notif_exp=time.time()+2.2
```

```
def ai_tick(self):
    if self.state!=S.PLAYING or self.mode!='bot' or self.cb.result: return
    if self.cb.turn==self.player_col: return

    if isinstance(self.ai, StockfishAI):
        # ── Stockfish path (already async) ──────────────────────────────
        if not self.ai_busy:
            self.ai.start_thinking(self.cb.fen())
            self.ai_busy=True
        else:
            result=self.ai.get_result()
            if result is not None:
                fr,fc,tr,tc2,sp,promo=result
                legal=self.cb.legal(fr,fc)
                matched=next((m for m in legal if m[0]==tr and m[1]==tc2),None)
                if matched is not None:
                    sp2=matched[2] if len(matched)>2 else sp
                    self._do(fr,fc,tr,tc2,sp2,promo)
                self.ai_busy=False
    else:
        # ── Fallback Python AI — run in background thread ────────────────
        if self.ai_busy: return
        self.ai_busy=True
        # snapshot board state for the thread
        cb_snap=self.ai._cl(self.cb)  # lightweight clone, no history
        def _bg():
            m=self.ai.best(cb_snap)
            self._py_ai_result=m
        self._py_ai_result=None
        t=threading.Thread(target=_bg,daemon=True); t.start()
        self._py_ai_thread=t

def _py_ai_poll(self):
    """Called every frame to check if the Python AI thread finished."""
    if not self.ai_busy: return
    if isinstance(self.ai, StockfishAI): return
    t=getattr(self,'_py_ai_thread',None)
    if t and not t.is_alive():
        m=getattr(self,'_py_ai_result',None)
        self.ai_busy=False
```

```
        if m and self.state==S.PLAYING and not self.cb.result:
            self._do(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None,'Q')

    # ── find move by san for record replay ──────────────────────────────
    def _find_san(self,cb,san):
        for m in cb.all_legal():
            fr,fc,tr,tc2=m[0],m[1],m[2],m[3]; sp=m[4] if len(m)>4 else None
            for pr in('Q','R','B','N'):
                if '=' in san and san[-1]!=pr: continue
                if cb._san(fr,fc,tr,tc2,sp,pr,cb.board)==san: return m
        return None

    def _open_rec_review(self,idx):
        recs=self.hist_recs
        if not recs or idx>=len(recs): return
        rec=recs[-(idx+1)]
        tmp=ChessBoard()
        for san in rec.get('moves',[]):
            m=self._find_san(tmp,san)
            if m is None: break
            tmp.make_move(m[0],m[1],m[2],m[3],m[4] if len(m)>4 else None)
        self.open_review(tmp.history[:],S.HISTORY)

    # ── event loop ──────────────────────────────────────────────────────
    def run(self):
        while True:
            self.clk.tick(60); self.tick()
            for ev in pygame.event.get():
                if ev.type==pygame.QUIT:
                    if isinstance(self.ai, StockfishAI): self.ai.close()
                    pygame.quit(); sys.exit()
                if ev.type==pygame.KEYDOWN:         self._key(ev.key)
                if ev.type==pygame.MOUSEBUTTONDOWN: self._click(ev.pos,ev.button)
                if ev.type==pygame.MOUSEBUTTONUP:   self._rel(ev.pos)
                if ev.type==pygame.MOUSEMOTION:
                    if self.drag_p: self.drag_pos=ev.pos
            if self.state==S.PLAYING: self.ai_tick(); self._py_ai_poll()
            self._draw()

    # ── key ─────────────────────────────────────────────────────────────
```

```
    def _key(self,key):
        if self.state==S.REVIEW:                                # FIX #4
            if key in(pygame.K_LEFT,pygame.K_a):  self.rev_idx=max(0,self.rev_idx-1)
            if key in(pygame.K_RIGHT,pygame.K_d):
self.rev_idx=min(len(self.rev_hist),self.rev_idx+1)
            if key==pygame.K_HOME: self.rev_idx=0
            if key==pygame.K_END:  self.rev_idx=len(self.rev_hist)
            if key==pygame.K_ESCAPE: self.state=self.rev_src
        elif self.state==S.PLAYING:
            if key==pygame.K_ESCAPE: self.state=S.MAIN
            if key==pygame.K_f:      self.flipped=not self.flipped
            if key==pygame.K_r:      self.open_review_game()
        elif self.state==S.HIST_CONFIRM:
            if key==pygame.K_ESCAPE: self.state=S.HISTORY
        elif key==pygame.K_ESCAPE: self.state=S.MAIN

    # ── click dispatcher ──────────────────────────────────────────────
    def _click(self,pos,btn):
        {S.MAIN:self._c_main,S.SETUP_BOT:self._c_sbot,S.SETUP_LOC:self._c_sloc,
         S.PLAYING:self._c_play,S.PROMOTION:self._c_promo,S.REVIEW:self._c_rev,
         S.HISTORY:self._c_hist,S.HIST_CONFIRM:self._c_hconf
        }.get(self.state,lambda p:None)(pos)

    def _rel(self,pos):
        if self.state not in(S.PLAYING,S.PROMOTION) or not self.drag_p: return
        x,y=pos; r,c=self.p2s(x,y)
        if r is not None and self.drag_fr:
            fr,fc=self.drag_fr
            if(fr,fc)!=(r,c): self.try_move(fr,fc,r,c)
        self.drag_p=None; self.drag_pos=None; self.drag_fr=None

    def _c_main(self,pos):
        x,y=pos; cx=WINDOW_W//2
        if cx-160<=x<=cx-10  and WINDOW_H//2-30<=y<=WINDOW_H//2+30: self.state=S.SETUP_BOT
        elif cx+10<=x<=cx+160 and WINDOW_H//2-30<=y<=WINDOW_H//2+30: self.state=S.SETUP_LOC
        elif cx-100<=x<=cx+100 and WINDOW_H//2+60<=y<=WINDOW_H//2+110: self.open_hist()

    def _c_sbot(self,pos):
        x,y=pos; cx=WINDOW_W//2
        # 11 difficulty buttons — same layout as _d_sbot
```

```
        bw,bh=78,48; row1=6; row2=5
        for i in range(11):
            if i<row1:
                total_w=row1*bw+(row1-1)*6
                bx=cx-total_w//2+i*(bw+6); by=222
            else:
                j=i-row1
                total_w=row2*bw+(row2-1)*6
                bx=cx-total_w//2+j*(bw+6); by=278
            if bx<=x<=bx+bw and by<=y<=by+bh: self.opt_ai=i+1
        # Play as buttons
        if cx-160<=x<=cx-20 and 362<=y<=402:  self.opt_col=WHITE
        elif cx+20<=x<=cx+160 and 362<=y<=402: self.opt_col=BLACK
        # Time controls
        for i in range(len(TIME_CONTROLS)):
            bx=cx-210+(i%3)*142; by=440+(i//3)*52
            if bx<=x<=bx+132 and by<=y<=by+42: self.opt_tc=i
        # Start
        if cx-100<=x<=cx+100 and 590<=y<=630: self.start('bot')
        # Back
        if 18<=x<=110 and 18<=y<=50: self.state=S.MAIN

    def _c_sloc(self,pos):
        x,y=pos; cx=WINDOW_W//2
        for i in range(len(TIME_CONTROLS)):
            bx=cx-210+(i%3)*142; by=262+(i//3)*52
            if bx<=x<=bx+132 and by<=y<=by+42: self.opt_tc=i
        if cx-100<=x<=cx+100 and 448<=y<=488: self.start('local')
        if 18<=x<=110 and 18<=y<=50: self.state=S.MAIN

    def _c_play(self,pos):
        x,y=pos
        if x>=BOARD_SIZE: self._c_panel(pos); return
        if self.cb.result: return
        r,c=self.p2s(x,y)
        if r is None: return
        if self.mode=='bot' and self.cb.turn!=self.player_col: return
        p=self.cb.board[r][c]
        if p and self.cb.col(p)==self.cb.turn:
            self.sel=(r,c); self.ltgts=self.cb.legal(r,c)
```

```
            self.drag_p=p; self.drag_pos=pos; self.drag_fr=(r,c)
        elif self.sel:
            self.try_move(self.sel[0],self.sel[1],r,c)

    def _c_panel(self,pos):
        x,y=pos; px=x-BOARD_SIZE
        if 558<=y<=588:
            if  10<=px<=95:  self.open_review_game()
            elif 105<=px<=190: self.start(self.mode)
            elif 200<=px<=285: self.state=S.MAIN

    def _c_promo(self,pos):
        x,y=pos; cx,cy=BOARD_SIZE//2,BOARD_SIZE//2
        for i,p in enumerate(['Q','R','B','N']):
            bx=cx-105+i*54; by=cy-22
            if bx<=x<=bx+52 and by<=y<=by+50:
                fr,fc,tr,tc2,sp=self.promo
                self._do(fr,fc,tr,tc2,sp,p)
                self.promo=None; self.state=S.PLAYING; return

    def _c_rev(self,pos):
        x,y=pos; ox=self.BOARD_OX; cx=ox+BOARD_SIZE//2; by0=BOARD_SIZE-46
        for bx,by,bw,bh,act in[(cx-132,by0,40,36,'first'),(cx-84,by0,40,36,'prev'),
                               (cx+44, by0,40,36,'next'),(cx+92,by0,40,36,'last')]:
            if bx<=x<=bx+bw and by<=y<=by+bh:
                if act=='first': self.rev_idx=0
                elif act=='prev': self.rev_idx=max(0,self.rev_idx-1)
                elif act=='next': self.rev_idx=min(len(self.rev_hist),self.rev_idx+1)
                elif act=='last': self.rev_idx=len(self.rev_hist)
                return
        panel_x=ox+BOARD_SIZE
        if x>=panel_x:
            pw=WINDOW_W-panel_x
            if 558<=y<=588 and panel_x+6<=x<=WINDOW_W-6:
                self.state=self.rev_src

    def _c_hist(self,pos):
        x,y=pos; cx=WINDOW_W//2
        if WINDOW_H-55<=y<=WINDOW_H-20 and cx-80<=x<=cx+80:
            self.state=S.MAIN; return
```

```
        item_h=70; list_y0=88
        if 100<=y<=WINDOW_H-80 and 30<=x<=WINDOW_W-30:
            idx=(y-list_y0)//item_h+self.hist_scroll
            if 0<=idx<len(self.hist_recs):
                self.hist_confirm=idx; self.state=S.HIST_CONFIRM  # FIX #5

    def _c_hconf(self,pos):
        x,y=pos; cx,cy=WINDOW_W//2,WINDOW_H//2
        if cx-120<=x<=cx-10 and cy+20<=y<=cy+65:  self._open_rec_review(self.hist_confirm)
        elif cx+10<=x<=cx+120 and cy+20<=y<=cy+65: self.state=S.HISTORY

    # ═══════════════════════════════════════════════════════════════════════
    # Drawing
    # ═══════════════════════════════════════════════════════════════════════
    def _draw(self):
        s=self.surf; s.fill(C_BG)
        if   self.state==S.MAIN:      self._d_main()
        elif self.state==S.SETUP_BOT: self._d_sbot()
        elif self.state==S.SETUP_LOC: self._d_sloc()
        elif self.state in(S.PLAYING,S.PROMOTION):
            self._d_play()
            if self.state==S.PROMOTION: self._d_promo()
        elif self.state==S.REVIEW:    self._d_rev()
        elif self.state==S.HISTORY:   self._d_hist()
        elif self.state==S.HIST_CONFIRM: self._d_hist(); self._d_hconf()
        pygame.display.flip()

    # ── Main menu ──────────────────────────────────────────────────────────
    def _d_main(self):
        s=self.surf; cx,cy=WINDOW_W//2,WINDOW_H//2; mx,my=pygame.mouse.get_pos()
        for r in range(8):
            for c in range(8):
                clr=(50,46,40) if(r+c)%2==0 else(38,35,30)

pygame.draw.rect(s,clr,(c*(WINDOW_W//8),r*(WINDOW_H//8),WINDOW_W//8,WINDOW_H//8))
        ov=pygame.Surface((WINDOW_W,WINDOW_H),pygame.SRCALPHA); ov.fill((14,13,11,218));
s.blit(ov,(0,0))
        tc(s,"CHESS",FNT_XL,(235,210,150),cx,cy-155)
        tc(s,"Full Rules  ·  10 AI Levels  ·  Time Controls",FNT_SM,C_TEXT2,cx,cy-112)
        pygame.draw.line(s,C_BORDER,(cx-230,cy-92),(cx+230,cy-92),1)
```

```
        for bx,by,bw,bh,lbl,clr in[(cx-160,cy-30,150,60,"vs Bot",C_BLUE),(cx+10,cy-
30,150,60,"Local 2P",C_ACCENT)]:
            hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
            c2=tuple(min(255,v+22) for v in clr) if hov else clr
            rr(s,c2,(bx,by,bw,bh),10,1,C_BORDER)
            tc(s,lbl,FNT_MDB,(10,10,10),bx+bw//2,by+bh//2)
        hbx,hby,hbw,hbh=cx-100,cy+60,200,50
        hov=(hbx<=mx<=hbx+hbw and hby<=my<=hby+hbh)
        rr(s,C_BTN_H if hov else C_BTN,(hbx,hby,hbw,hbh),8,1,C_BORDER)
        tc(s,"Game Records",FNT_MD,C_TEXT,hbx+hbw//2,hby+hbh//2)
        tc(s,"F: flip  |  R: review  |  ESC: menu",FNT_XS,C_TEXT3,cx,WINDOW_H-18)

    # ── Setup screens ──────────────────────────────────────────────────
    def _d_sbot(self):
        s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
        tc(s,"Play vs Bot",FNT_LG,C_TEXT,cx,42)
        rr(s,C_BTN,(18,18,90,32),5,1,C_BORDER); tc(s,"< Back",FNT_SM,C_TEXT2,63,34)
        tc(s,"AI Difficulty",FNT_MD,C_TEXT2,cx,196)
        names=["Beginner","Novice","Amateur","Casual","Intermediate",
               "Advanced","Expert","Master","IM","GM","Super GM"]
        elos =[500,800,1200,1600,1800,2000,2200,2300,2400,2500,2700]
        # 11 buttons: first row 6, second row 5 — centred
        row1=6; row2=5
        bw,bh=78,48
        for i in range(11):
            if i<row1:
                total_w=row1*bw+(row1-1)*6
                bx=cx-total_w//2+i*(bw+6); by=222
            else:
                j=i-row1
                total_w=row2*bw+(row2-1)*6
                bx=cx-total_w//2+j*(bw+6); by=278
            sel=(self.opt_ai==i+1)
            c=C_BTN_A if sel else(C_BTN_H if(bx<=mx<=bx+bw and by<=my<=by+bh) else C_BTN)
            rr(s,c,(bx,by,bw,bh),7,1,C_BORDER)
            tc(s,str(i+1),FNT_SMB,(255,255,255) if sel else C_TEXT2,bx+bw//2,by+12)
            tc(s,names[i],FNT_XS,(255,255,255) if sel else C_TEXT3,bx+bw//2,by+27)
            tc(s,str(elos[i]),FNT_XS,C_GOLD if sel else C_TEXT3,bx+bw//2,by+39)
        tc(s,"Play as",FNT_MD,C_TEXT2,cx,348)
        for lbl,col2,bx in[("White",WHITE,cx-160),("Black",BLACK,cx+20)]:
```

```
        bw2,bh2=140,40; by=362; sel=(self.opt_col==col2)
        c=C_BTN_A if sel else(C_BTN_H if(bx<=mx<=bx+bw2 and by<=my<=by+bh2) else C_BTN)
        rr(s,c,(bx,by,bw2,bh2),7,1,C_BORDER)
        tc(s,lbl,FNT_MDB,(255,255,255) if sel else C_TEXT,bx+bw2//2,by+bh2//2)
    tc(s,"Time Control",FNT_MD,C_TEXT2,cx,420)
    self._d_tc(s,cx,mx,my,440)
    sbx,sby,sbw,sbh=cx-100,590,200,40; hov=(sbx<=mx<=sbx+sbw and sby<=my<=sby+sbh)
    rr(s,C_ACCENT if hov else(72,148,72),(sbx,sby,sbw,sbh),8,1,C_BORDER)
    tc(s,"Start Game",FNT_MDB,(10,30,10),sbx+sbw//2,sby+sbh//2)

def _d_sloc(self):
    s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
    tc(s,"Local 2-Player",FNT_LG,C_TEXT,cx,42)
    rr(s,C_BTN,(18,18,90,32),5,1,C_BORDER); tc(s,"< Back",FNT_SM,C_TEXT2,63,34)
    tc(s,"Time Control",FNT_MD,C_TEXT2,cx,238)
    self._d_tc(s,cx,mx,my,258)
    sbx,sby,sbw,sbh=cx-100,448,200,40; hov=(sbx<=mx<=sbx+sbw and sby<=my<=sby+sbh)
    rr(s,C_ACCENT if hov else(72,148,72),(sbx,sby,sbw,sbh),8,1,C_BORDER)
    tc(s,"Start Game",FNT_MDB,(10,30,10),sbx+sbw//2,sby+sbh//2)

def _d_tc(self,s,cx,mx,my,y0):
    for i,t2 in enumerate(TIME_CONTROLS):
        bx=cx-210+(i%3)*142; by=y0+(i//3)*52; bw,bh=132,42; sel=(self.opt_tc==i)
        hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
        c=C_BTN_A if sel else(C_BTN_H if hov else C_BTN)
        rr(s,c,(bx,by,bw,bh),7,1,C_BORDER)
        tc(s,t2['label'],FNT_MDB,(255,255,255) if sel else C_TEXT,bx+bw//2,by+14)
        tc(s,t2['name'], FNT_XS, (210,210,210) if sel else C_TEXT3,bx+bw//2,by+30)

# ── Playing ──────────────────────────────────────────────────────────
def _d_play(self):
    self._d_board(self.surf)
    self._d_pieces(self.surf,self.cb.board if self.cb else None)
    self._d_panel(self.surf)
    self._d_drag(self.surf)
    if self.cb and self.cb.result: self._d_result(self.surf)

def _d_board(self,s,lm=None,sel=None,tgts=None):
    fl=self.flipped
    lm  =lm   if lm   is not None else self.lm
```

```
        sel =sel  if sel  is not None else self.sel
        tgts=tgts if tgts is not None else self.ltgts
        chk_sq=None
        if self.cb and self.cb.is_check() and not self.cb.result:
            king='K' if self.cb.turn==WHITE else 'k'
            for r in range(8):
                for c in range(8):
                    if self.cb.board[r][c]==king: chk_sq=(r,c)
        for r in range(8):
            for c in range(8):
                px=(7-c)*SQ if fl else c*SQ; py=(7-r)*SQ if fl else r*SQ
                base=C_LIGHT_SQ if(r+c)%2==0 else C_DARK_SQ
                pygame.draw.rect(s,base,(px,py,SQ,SQ))
                if lm and((r,c)==lm[0] or(r,c)==lm[1]):
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*C_HIGHLIGHT,170));
s.blit(hl,(px,py))
                if sel and(r,c)==sel:
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA);
hl.fill((*C_SELECTED[:3],210)); s.blit(hl,(px,py))
                if chk_sq and(r,c)==chk_sq:
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*C_CHECK,185));
s.blit(hl,(px,py))
        cb_b=self.cb.board if self.cb else [[None]*8 for _ in range(8)]
        for m in tgts:
            tr2,tc2=m[0],m[1]; px=(7-tc2)*SQ if fl else tc2*SQ; py=(7-tr2)*SQ if fl else
tr2*SQ
            hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA)
            if cb_b[tr2][tc2]: pygame.draw.circle(hl,(0,0,0,72),(SQ//2,SQ//2),SQ//2-2,5)
            else:               pygame.draw.circle(hl,(0,0,0,72),(SQ//2,SQ//2),SQ//6)
            s.blit(hl,(px,py))
        for i in range(8):
            ci=7-i if fl else i; ri=i if fl else 7-i
            lc=C_DARK_SQ if(7+i)%2==0 else C_LIGHT_SQ
            lr=C_DARK_SQ if i%2==1    else C_LIGHT_SQ
            lt=FNT_XS.render('abcdefgh'[ci],True,lc); s.blit(lt,(i*SQ+SQ-lt.get_width()-
3,BOARD_SIZE-lt.get_height()-2))
            ln=FNT_XS.render(str(ri+1),True,lr);      s.blit(ln,(3,i*SQ+2))

    def _d_pieces(self,s,board,fl=None):
        if board is None: return
```

```
        if fl is None: fl=self.flipped
        df=self.drag_fr
        for r in range(8):
            for c in range(8):
                if df and(r,c)==df: continue
                p=board[r][c]
                if not p: continue
                px=(7-c)*SQ if fl else c*SQ; py=(7-r)*SQ if fl else r*SQ
                draw_piece_at(s,p,px,py)

    def _d_drag(self,s):
        if self.drag_p and self.drag_pos:
            x,y=self.drag_pos; draw_piece_at(s,self.drag_p,x-SQ//2,y-SQ//2)

    def _d_result(self,s):
        ov=pygame.Surface((BOARD_SIZE,78),pygame.SRCALPHA); ov.fill((18,17,15,215))
        s.blit(ov,(0,BOARD_SIZE//2-39))
        r=self.cb.result
        msg,clr=("White wins",C_ACCENT) if r==WHITE else("Black wins",C_RED) if r==BLACK
else("Draw",C_TEXT2)
        tc(s,msg,FNT_LG,clr,BOARD_SIZE//2,BOARD_SIZE//2-11)
        tc(s,self.cb.result_reason.capitalize(),FNT_SM,C_TEXT2,BOARD_SIZE//2,BOARD_SIZE//2+16)

    # ── helpers for captured pieces ──────────────────────────────────────
    def _captured(self):
        """Return (white_captured, black_captured, advantage_color, adv_pts).
        white_captured = pieces white has taken (i.e. black pieces removed from board).
        black_captured = pieces black has taken (i.e. white pieces removed from board)."""
        PV = {'P':1,'N':3,'B':3,'R':5,'Q':9}
        start = {'P':8,'N':2,'B':2,'R':2,'Q':1}
        on_board = {}
        for r in range(8):
            for c in range(8):
                p = self.cb.board[r][c]
                if p: on_board[p] = on_board.get(p,0)+1
        # pieces white captured = missing black pieces
        w_cap=[]
        for pt,cnt in start.items():
            missing = cnt - on_board.get(pt.lower(),0)
            w_cap.extend([pt]*missing)
```

```
        # pieces black captured = missing white pieces
        b_cap=[]
        for pt,cnt in start.items():
            missing = cnt - on_board.get(pt,0)
            b_cap.extend([pt]*missing)
        ws = sum(PV.get(p,0) for p in w_cap)
        bs = sum(PV.get(p,0) for p in b_cap)
        adv = ws-bs
        return w_cap, b_cap, adv

    def _draw_caps(self, s, caps, score_adv, x, y, row_w):
        """Draw captured piece symbols in a compact row."""
        PV={'P':1,'N':3,'B':3,'R':5,'Q':9}
        # sort by value
        caps_sorted = sorted(caps, key=lambda p: PV.get(p,0))
        font, use_uni = _pfont(14)
        cx2 = x
        for p in caps_sorted:
            sym = UNICODE_SYMS.get(p.lower(),'?') if use_uni else p
            t = font.render(sym, True, C_TEXT2)
            s.blit(t,(cx2, y)); cx2 += t.get_width()+1
            if cx2 > x+row_w-20: break   # don't overflow
        if score_adv > 0:
            adv_t = FNT_XS.render(f"+{score_adv}", True, C_ACCENT)
            s.blit(adv_t,(cx2+4, y+1))

    # ── Panel ──────────────────────────────────────────────────────────
    def _d_panel(self,s):
        px0=BOARD_SIZE; PW=PANEL_WIDTH; W=PW-28; x=px0+14; mx,my=pygame.mouse.get_pos()
        pygame.draw.rect(s,C_PANEL,(px0,0,PW,WINDOW_H))
        pygame.draw.line(s,C_BORDER,(px0,0),(px0,WINDOW_H),1)
        y=10

        # ── Header: mode + ELO ──────────────────────────────────────────
        t2=TIME_CONTROLS[self.opt_tc]
        if self.mode=='bot':
            elo = StockfishAI.LEVEL_ELO.get(self.opt_ai, '?')
            lbl = f"vs Bot · Lv{self.opt_ai} · {elo} Elo · {t2['label']}"
        else:
            lbl = f"Local 2P · {t2['label']}"
```

```
        tc(s,lbl,FNT_XS,C_TEXT2,px0+PW//2,y+7); y+=18

        # ── Captured pieces + material advantage ──────────────────────────
        w_cap, b_cap, adv = self._captured()
        # adv > 0 → white ahead,  adv < 0 → black ahead

        # Determine which player is "opponent" (top) and "player" (bottom)
        # If playing as black: top=black(player), bottom=white(opponent)
        # Default / white / local: top=black(opponent), bottom=white(player)
        player_is_black = (self.mode=='bot' and self.player_col==BLACK)

        if player_is_black:
            # top = black (player), bottom = white (opponent)
            top_color, bot_color = BLACK, WHITE
            top_time,  bot_time  = self.bt, self.wt
            top_label, bot_label = "Black (You)", "White (Bot)"
            top_cap,   bot_cap   = b_cap, w_cap   # black captured whites; white captured
blacks
            top_adv  = max(0,-adv)   # black advantage
            bot_adv  = max(0, adv)   # white advantage
        else:
            top_color, bot_color = BLACK, WHITE
            top_time,  bot_time  = self.bt, self.wt
            if self.mode=='bot':
                top_label = "Black (Bot)"; bot_label = "White (You)"
            else:
                top_label = "Black"; bot_label = "White"
            top_cap,   bot_cap   = b_cap, w_cap
            top_adv  = max(0,-adv)
            bot_adv  = max(0, adv)

        # ── TOP player box ────────────────────────────────────────────────
        top_act = (self.cb.turn==top_color and self.clk_run and not self.cb.result)
        rr(s,(52,50,44) if top_act else C_PANEL2,(x,y,W,44),6,1,C_BORDER)
        tc(s,top_label,FNT_XS,C_TEXT2,x+W//2,y+10)
        tc(s,fmt(top_time),FNT_CLK,C_GOLD if top_act else C_TEXT2,x+W//2,y+30)
        y+=50

        # Pieces captured BY top player = opponent's pieces they took
        # top captures opponent pieces: if top=BLACK → took white pieces (uppercase)
```

```
        #                               if top=WHITE → took black pieces (lowercase)
        top_caps_disp = w_cap if top_color==BLACK else b_cap   # white pieces BLACK took /
black pieces WHITE took
        top_adv_pts   = max(0, -adv) if top_color==BLACK else max(0, adv)
        if top_caps_disp:
            self._draw_caps(s, top_caps_disp, top_adv_pts, x, y, W)
        y+=16

        pygame.draw.line(s,C_SEP,(x,y),(x+W,y),1); y+=8

        # ── Move list ──────────────────────────────────────────────────────
        hist=self.cb.history; ROW=17; max_vis=13
        pairs=[]
        i=0
        while i<len(hist):
            ws=hist[i]['move']['san']; i+=1
            bs=hist[i]['move']['san'] if i<len(hist) else ''; i+=1
            pairs.append((len(pairs)+1,ws,bs))
        start_p=max(0,len(pairs)-max_vis); ml_y=y
        for rel,(mn,ws,bs) in enumerate(pairs[start_p:]):
            ry=ml_y+rel*ROW
            tl(s,f"{mn}.",FNT_MONO_SM,C_TEXT3,x,ry)
            tl(s,ws,      FNT_MONO_SM,(225,220,205),x+30,ry)
            if bs: tl(s,bs,FNT_MONO_SM,(205,205,220),x+118,ry)
        y=ml_y+max_vis*ROW+4

        pygame.draw.line(s,C_SEP,(x,y),(x+W,y),1); y+=8
        if self.cb.is_check() and not self.cb.result:
            tc(s,"Check!",FNT_SMB,C_RED,px0+PW//2,y+8); y+=22
        if self.ai_busy:
            sf=isinstance(self.ai,StockfishAI) and self.ai.ready
            lbl2="Stockfish thinking..." if sf else "AI thinking..."
            tc(s,lbl2,FNT_SM,C_GOLD,px0+PW//2,y+8); y+=20
        if self.notif and time.time()<self.notif_exp:
            tc(s,self.notif,FNT_SM,C_ACCENT,px0+PW//2,y+8)

        # ── BOTTOM captured pieces ─────────────────────────────────────────
        bot_caps_disp = b_cap if bot_color==BLACK else w_cap   # symmetric
        bot_adv_pts   = max(0, -adv) if bot_color==BLACK else max(0, adv)
        bot_cap_y = 488
```

```
        if bot_caps_disp:
            self._draw_caps(s, bot_caps_disp, bot_adv_pts, x, bot_cap_y, W)

        # ── BOTTOM player box ──────────────────────────────────────────
        bot_act = (self.cb.turn==bot_color and self.clk_run and not self.cb.result)
        rr(s,(52,50,44) if bot_act else C_PANEL2,(x,506,W,44),6,1,C_BORDER)
        tc(s,bot_label,FNT_XS,C_TEXT2,x+W//2,506+10)
        tc(s,fmt(bot_time),FNT_CLK,C_GOLD if bot_act else C_TEXT,x+W//2,506+30)

        # ── Buttons ────────────────────────────────────────────────────
        for lbl2,bx,by,bw2,bh,c in[("Review",10,558,84,30,C_GOLD),
                                   ("Rematch",104,558,80,30,C_BTN),
                                   ("Menu",194,558,84,30,C_BTN)]:
            ax=px0+bx; hov=(ax<=mx<=ax+bw2 and by<=my<=by+bh)
            rr(s,C_BTN_H if hov else c,(ax,by,bw2,bh),5,1,C_BORDER)
            tc(s,lbl2,FNT_SM,C_TEXT,ax+bw2//2,by+bh//2)
        tc(s,"F:flip  R:review  ESC:menu",FNT_XS,C_TEXT3,px0+PW//2,WINDOW_H-10)

    def _d_promo(self):
        s=self.surf; cx,cy=BOARD_SIZE//2,BOARD_SIZE//2; mx,my=pygame.mouse.get_pos()
        ov=pygame.Surface((BOARD_SIZE,BOARD_SIZE),pygame.SRCALPHA); ov.fill((0,0,0,168));
s.blit(ov,(0,0))
        rr(s,(46,44,38),(cx-125,cy-60,250,118),12,1,C_BORDER)
        tc(s,"Promote to:",FNT_MD,C_TEXT2,cx,cy-42)
        for i,p in enumerate(['Q','R','B','N']):
            piece=p if self.cb.turn==WHITE else p.lower()
            bx=cx-105+i*54; by=cy-22; hov=(bx<=mx<=bx+52 and by<=my<=by+50)
            rr(s,C_BTN_H if hov else C_BTN,(bx,by,52,50),7,1,C_BORDER)
            draw_piece_at(s,piece,bx,by,50)

    # ── Review ─────────────────────────────────────────────────────────
    # Layout in review mode:
    #   x=0..18   : eval bar (white=bottom, black=top)
    #   x=18..658 : chess board (640px)  → board offset = 18
    #   x=658..958: panel (300px)
    EVAL_BAR_W  = 18
    BOARD_OX    = 18   # board x-offset in review mode

    def _d_rev(self):
        s=self.surf; idx=self.rev_idx
```

```
        board=self.rev_boards[idx] if idx<len(self.rev_boards) else (self.rev_boards[-1] if
self.rev_boards else None)
        lm=None
        if idx>0: m=self.rev_hist[idx-1]['move']; lm=(m['from'],m['to'])

        # ── Eval bar (left strip) ──────────────────────────────────────
        self._d_eval_bar(s)

        # ── Board (shifted right by EVAL_BAR_W) ────────────────────────
        # Draw squares
        fl=self.flipped
        chk_sq=None  # no check highlight in review
        for r in range(8):
            for c in range(8):
                px=self.BOARD_OX+((7-c)*SQ if fl else c*SQ)
                py=(7-r)*SQ if fl else r*SQ
                base=C_LIGHT_SQ if(r+c)%2==0 else C_DARK_SQ
                pygame.draw.rect(s,base,(px,py,SQ,SQ))
                if lm and ((r,c)==lm[0] or (r,c)==lm[1]):
                    hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*C_HIGHLIGHT,170));
s.blit(hl,(px,py))
        # Classification highlight on destination square
        if idx>0:
            clf=self.rev_analyser.get_classif(idx) if self.rev_analyser else 'none'
            clf_clr,_,_=CLF_INFO.get(clf,CLF_INFO['none'])
            tr2,tc3=lm[1]
            px2=self.BOARD_OX+((7-tc3)*SQ if fl else tc3*SQ)
            py2=(7-tr2)*SQ if fl else tr2*SQ
            hl=pygame.Surface((SQ,SQ),pygame.SRCALPHA); hl.fill((*clf_clr,90));
s.blit(hl,(px2,py2))
            # Badge circle on corner
            bx3=px2+(SQ-18 if not fl else 0); by3=py2
            pygame.draw.circle(s,clf_clr,(bx3+9,by3+9),9)
            sym=CLF_INFO.get(clf,('','',''))[1]
            if sym:
                f2,_=_pfont(10); st=f2.render(sym,True,(255,255,255))
                s.blit(st,(bx3+9-st.get_width()//2,by3+9-st.get_height()//2))
        # Coordinates
        for i in range(8):
            ci=7-i if fl else i; ri=i if fl else 7-i
```

```
            lc=C_DARK_SQ if(7+i)%2==0 else C_LIGHT_SQ
            lr=C_DARK_SQ if i%2==1    else C_LIGHT_SQ
            lt=FNT_XS.render('abcdefgh'[ci],True,lc)
            s.blit(lt,(self.BOARD_OX+i*SQ+SQ-lt.get_width()-3,BOARD_SIZE-lt.get_height()-2))
            ln=FNT_XS.render(str(ri+1),True,lr)
            s.blit(ln,(self.BOARD_OX+3,i*SQ+2))
        # Pieces
        if board:
            for r in range(8):
                for c in range(8):
                    p=board[r][c]
                    if not p: continue
                    px=self.BOARD_OX+((7-c)*SQ if fl else c*SQ)
                    py=(7-r)*SQ if fl else r*SQ
                    draw_piece_at(s,p,px,py)

        self._d_rev_nav(s)
        self._d_rev_panel(s)

    def _d_eval_bar(self,s):
        """Draw a vertical eval bar on the left edge."""
        ra=self.rev_analyser; idx=self.rev_idx
        ev=None
        if ra: ev=ra.get_eval(idx)
        BW=self.EVAL_BAR_W; H=BOARD_SIZE
        # Background
        pygame.draw.rect(s,(30,28,24),(0,0,BW,H))
        # Convert eval to 0..1 (white fraction of bar, measured from BOTTOM)
        if ev is None:
            wfrac=0.5
        else:
            # clamp to ±1000cp, sigmoid-ish
            clamped=max(-1000,min(1000,ev))
            wfrac=0.5+clamped/2000.0
        wfrac=max(0.05,min(0.95,wfrac))
        black_h=int(H*(1-wfrac)); white_h=H-black_h
        # Black portion (top)
        pygame.draw.rect(s,(30,30,30),(0,0,BW,black_h))
        # White portion (bottom)
        pygame.draw.rect(s,(220,215,200),(0,black_h,BW,white_h))
```

```
        # Midline
        pygame.draw.line(s,(80,78,72),(0,H//2),(BW,H//2),1)
        # Score text
        if ev is not None:
            if abs(ev)>=29000: txt="M" if ev>0 else "-M"
            else: txt=f"{ev/100:+.1f}" if abs(ev)<1000 else f"{ev/100:+.0f}"
            fs=10; f2,_=_pfont(fs)
            st=FNT_XS.render(txt,True,(200,200,200) if abs(ev)<50 else (C_ACCENT if ev>0 else
C_RED))
            # rotate 90° for vertical text
            st_r=pygame.transform.rotate(st,90)
            s.blit(st_r,(BW//2-st_r.get_width()//2, H//2-st_r.get_height()//2))
        # "Analysing..." if not done
        if ra and not ra.done:
            dot_y=H-24
            dt=FNT_XS.render("…",True,C_GOLD)
            s.blit(dt,(BW//2-dt.get_width()//2,dot_y))

    def _d_rev_nav(self,s):
        ox=self.BOARD_OX; cx=ox+BOARD_SIZE//2; by0=BOARD_SIZE-46; mx,my=pygame.mouse.get_pos()
        for bx,by,bw,bh,lbl in[(cx-132,by0,40,36,'|<'),(cx-84,by0,40,36,'<'),
                               (cx+44, by0,40,36,'>'),(cx+92,by0,40,36,'|>')]:
            hov=(bx<=mx<=bx+bw and by<=my<=by+bh)
            rr(s,C_BTN_H if hov else C_PANEL3,(bx,by,bw,bh),6,1,C_BORDER)
            tc(s,lbl,FNT_MDB,C_TEXT,bx+bw//2,by+bh//2)
        tc(s,f"{self.rev_idx} / {len(self.rev_hist)}",FNT_SM,C_TEXT2,cx,by0+18)
        tc(s,"Arrow keys or buttons to navigate",FNT_XS,C_TEXT3,cx,BOARD_SIZE-7)

    def _d_rev_panel(self,s):
        # Panel starts at BOARD_OX + BOARD_SIZE
        px0=self.BOARD_OX+BOARD_SIZE; PW=PANEL_WIDTH-(self.BOARD_OX); x=px0+14
        mx,my=pygame.mouse.get_pos()
        pygame.draw.rect(s,C_PANEL,(px0,0,WINDOW_W-px0,WINDOW_H))
        pygame.draw.line(s,C_BORDER,(px0,0),(px0,WINDOW_H),1)
        W=WINDOW_W-px0-28; y=14
        tc(s,"Game Review",FNT_LG,C_GOLD,px0+(WINDOW_W-px0)//2,y+10); y+=34

        # ── Analysis status / current move feedback ──────────────────────
        ra=self.rev_analyser; idx=self.rev_idx
        if idx>0 and ra:
```

```
            clf=ra.get_classif(idx)
            clf_clr,sym,desc=CLF_INFO.get(clf,CLF_INFO['none'])
            if clf!='none':
                rr(s,(*clf_clr,40),(x,y,W,38),6)  # won't work with alpha directly

pygame.draw.rect(s,(clf_clr[0]//4,clf_clr[1]//4,clf_clr[2]//4),(x,y,W,38),border_radius=6)
                pygame.draw.rect(s,clf_clr,(x,y,W,38),1,border_radius=6)
                tc(s,f"{sym}  {clf.upper()}",FNT_SMB,clf_clr,x+W//2,y+12)
                tc(s,desc.split('—')[1].strip() if '—' in desc else
desc,FNT_XS,C_TEXT2,x+W//2,y+27)
                y+=46
        elif ra and not ra.done:
            tc(s,"Analysing game...",FNT_SM,C_GOLD,px0+(WINDOW_W-px0)//2,y+10); y+=24
        else:
            y+=4

        pygame.draw.line(s,C_SEP,(x,y),(x+W,y),1); y+=8

        # ── Paired move list with classification badges ──────────────────────
        hist=self.rev_hist; cur=self.rev_idx; ROW=18; max_vis=16
        pairs=[]
        i=0
        while i<len(hist):
            ws=hist[i]['move']['san']; wclf=ra.get_classif(i+1) if ra else 'none'; i+=1
            bs=hist[i]['move']['san'] if i<len(hist) else ''
            bclf=ra.get_classif(i+1) if (ra and i<len(hist)) else 'none'; i+=1
            pairs.append((len(pairs)+1, ws,wclf, bs,bclf))
        cur_row=(cur-1)//2 if cur>0 else 0
        start=max(0,cur_row-max_vis//2); end=min(len(pairs),start+max_vis); start=max(0,end-
max_vis)

        COL_NUM=x; COL_W=x+28; COL_B=x+W//2+4
        for rel,(mn,ws,wclf,bs,bclf) in enumerate(pairs[start:end]):
            pi=start+rel; ry=y+rel*ROW
            w_act=(cur==pi*2+1); b_act=(cur==pi*2+2)
            if w_act: pygame.draw.rect(s,(68,65,48),(px0+4,ry-1,W//2,ROW),border_radius=3)
            if b_act: pygame.draw.rect(s,(68,65,48),(px0+4+W//2,ry-
1,W//2,ROW),border_radius=3)
            tl(s,f"{mn}.",FNT_MONO_SM,C_TEXT3,COL_NUM,ry)
            # White move + badge
```

```
            wclr_b,wsym,_=CLF_INFO.get(wclf,CLF_INFO['none'])
            wclr=C_GOLD if w_act else (wclr_b if wclf not in('none','best') else
(225,220,205))
            tl(s,ws,FNT_MONO_SM,wclr,COL_W,ry)
            if wsym and wclf not in('none',):
                bt=FNT_XS.render(wsym,True,wclr_b)
                s.blit(bt,(COL_W+40,ry+1))
            # Black move + badge
            if bs:
                bclr_b,bsym,_=CLF_INFO.get(bclf,CLF_INFO['none'])
                bclr=C_GOLD if b_act else (bclr_b if bclf not in('none','best') else
(205,205,220))
                tl(s,bs,FNT_MONO_SM,bclr,COL_B,ry)
                if bsym and bclf not in('none',):
                    bt2=FNT_XS.render(bsym,True,bclr_b)
                    s.blit(bt2,(COL_B+40,ry+1))

        bx,by2,bw,bh=px0+6,558,WINDOW_W-px0-12,30
        hov=(bx<=mx<=bx+bw and by2<=my<=by2+bh)
        rr(s,C_BTN_H if hov else C_BTN,(bx,by2,bw,bh),5,1,C_BORDER)
        back="< Back to Game" if self.rev_src==S.PLAYING else "< Back to Records"
        tc(s,back,FNT_SM,C_TEXT,px0+(WINDOW_W-px0)//2,by2+bh//2)

    # ── History screen ──────────────────────────────────────────────────
    def _d_hist(self):
        s=self.surf; cx=WINDOW_W//2; mx,my=pygame.mouse.get_pos()
        tc(s,"Game Records",FNT_LG,C_TEXT,cx,45)
        pygame.draw.line(s,C_BORDER,(40,70),(WINDOW_W-40,70),1)
        recs=self.hist_recs
        if not recs: tc(s,"No games recorded yet.",FNT_MD,C_TEXT2,cx,WINDOW_H//2)
        else:
            item_h=70; list_y0=88; vis=min(7,(WINDOW_H-170)//item_h)
            for rel in range(vis):
                idx=rel+self.hist_scroll
                if idx>=len(recs): break
                rec=recs[-(idx+1)]; ry=list_y0+rel*item_h
                hov=(30<=mx<=WINDOW_W-30 and ry<=my<=ry+item_h-3)
                bg=C_PANEL3 if hov else(C_PANEL2 if rel%2==0 else C_PANEL)
                rr(s,bg,(30,ry,WINDOW_W-60,item_h-4),6,1,C_BORDER)
                res=rec.get('result','?')
```

```
                rclr=C_ACCENT if'White' in res else(C_RED if'Black' in res else C_TEXT2)
                tl(s,res,FNT_MDB,rclr,50,ry+6)
                reason=rec.get('reason','').capitalize()
                if reason: tl(s,f"by {reason}",FNT_XS,C_TEXT2,50,ry+24)
                mode='vs Bot' if rec.get('mode')=='bot' else'Local 2P'
                lvl=f" · Lv{rec.get('ai_level','?')}" if rec.get('mode')=='bot' else''
                tl(s,f"{mode}{lvl}  ·  {rec.get('tc','?')}",FNT_SM,C_TEXT,230,ry+8)
                tl(s,f"{rec.get('total_moves','?')} moves",FNT_XS,C_TEXT2,230,ry+26)
                tl(s,rec.get('date',''),FNT_XS,C_TEXT3,WINDOW_W-215,ry+8)
                moves=rec.get('moves',[]); prev=' '.join(f"{(i//2)+1}.{m}" if i%2==0 else m
for i,m in enumerate(moves[:10]))
                if len(moves)>10: prev+='...'
                tl(s,prev,FNT_XS,C_TEXT3,50,ry+44)
                if hov: tc(s,"Click to review",FNT_XS,C_GOLD,WINDOW_W-90,ry+item_h//2-4)
            total=len(recs)
            if total>vis:
                area=WINDOW_H-170; sbh=max(20,int(vis/total*area))
                sby=list_y0+int(self.hist_scroll/max(1,total-vis)*(area-sbh))
                pygame.draw.rect(s,C_PANEL3,(WINDOW_W-14,list_y0,6,area))
                pygame.draw.rect(s,C_TEXT2,(WINDOW_W-14,sby,6,sbh),border_radius=3)
        bbx,bby,bbw,bbh=cx-80,WINDOW_H-48,160,34; hov=(bbx<=mx<=bbx+bbw and bby<=my<=bby+bbh)
        rr(s,C_BTN_H if hov else C_BTN,(bbx,bby,bbw,bbh),6,1,C_BORDER)
        tc(s,"< Main Menu",FNT_SM,C_TEXT,cx,bby+bbh//2)

    # FIX #5 — confirm overlay
    def _d_hconf(self):
        s=self.surf; cx,cy=WINDOW_W//2,WINDOW_H//2; mx,my=pygame.mouse.get_pos()
        ov=pygame.Surface((WINDOW_W,WINDOW_H),pygame.SRCALPHA); ov.fill((0,0,0,155));
s.blit(ov,(0,0))
        rr(s,(46,44,38),(cx-185,cy-80,370,170),12,1,C_BORDER)
        tc(s,"Review this game?",FNT_LG,C_TEXT,cx,cy-52)
        if self.hist_confirm is not None and self.hist_recs:
            rec=self.hist_recs[-(self.hist_confirm+1)]
            info=f"{rec.get('result','?')}  ·  {rec.get('total_moves','?')} moves  ·
{rec.get('date','')}"
            tc(s,info,FNT_SM,C_TEXT2,cx,cy-22)
        for lbl2,bx,c in[("Review >",cx-120,C_BLUE),("Cancel",cx+10,C_BTN)]:
            bw2,bh2=110,46; by=cy+18; hov=(bx<=mx<=bx+bw2 and by<=my<=by+bh2)
            rr(s,tuple(min(255,v+20) for v in c) if hov else c,(bx,by,bw2,bh2),8,1,C_BORDER)
            tc(s,lbl2,FNT_MDB,C_TEXT,bx+bw2//2,by+bh2//2)
```

```
# ═══════════════════════════════════════════════════════════════════════════

if __name__ == '__main__':
    Game().run()
```

Revision #1
Created 2026-03-23 07:10:30 UTC by Samuel Lee
Updated 2026-03-23 07:11:14 UTC by Samuel Lee