Manual Completo de PicoTutor

Índice

Introducción

PicoTutor es el sistema de análisis y entrenamiento integrado en PicoChess que actúa como un entrenador de ajedrez inteligente. Su función principal es evaluar las jugadas del usuario en tiempo real, proporcionar sugerencias de mejora y ofrecer análisis detallados de las posiciones.

Características Principales:

- **Análisis en tiempo real** de las jugadas del usuario
- **Evaluación automática** con símbolos estándar (!, ?, !!, ??, !?, ?!)
- **Sugerencias de mejores jugadas** (hints)
- **Análisis de aperturas** con identificación ECO
- **Soporte para partidas PGN** paso a paso
- **Sistema de comentarios** contextuales
- **Múltiples modos de análisis** (profundo y obvio)

Arquitectura y Componentes

Estructura Principal

PicoTutor utiliza una arquitectura asíncrona basada en dos motores de análisis:

`` PicoTutor ├── best_engine (Motor Profundo) │ ├── Profundidad: 17 niveles │ ├── MultiPV: 50 variantes │ └── Threads: 1-2 hilos ├── obvious_engine (Motor Obvio) │ ├── Profundidad: 5 niveles │ ├── MultiPV: 50 variantes │ └── Threads: 1 hilo └── Sistemas de Soporte ├── Análisis de Aperturas ├── Gestión de Comentarios ├── Evaluación de Jugadas └── Sincronización de Tableros `

Componentes Clave

1. Motores de Análisis

- **best_engine**: Motor principal para análisis profundo (17 ply)
- **obvious_engine**: Motor secundario para análisis rápido (5 ply)

2. Estructuras de Datos

`python

Información de análisis por color

best_info = {chess.WHITE: [], chess.BLACK: []} obvious_info = {chess.WHITE: [], chess.BLACK: []}

Jugadas evaluadas por color

```
best_moves = {chess.WHITE: [], chess.BLACK: []} obvious_moves =
{chess.WHITE: [], chess.BLACK: []}
```

Historial de jugadas del usuario

best_history = {chess.WHITE: [], chess.BLACK: []} obvious_history = {chess.WHITE: [], chess.BLACK: []} `

3. Sistema de Evaluación

- **Evaluación de jugadas**: Comparación con las mejores opciones
- **Cálculo de pérdida de centipeones** (CPL - Centipawn Loss)
- **Detección de errores** según umbrales configurables
- **Generación de comentarios** automáticos

Inicialización y Configuración

Constructor Principal

`python def __init__(self, i_ucishell, i_engine_path, i_player_color, i_fen, i_comment_file, i_lang, i_always_run_tutor, loop): `

Parámetros de Inicialización:

Parámetro	Tipo	Descripción	Valor por Defecto
i_ucishell	UciShell	Interfaz de comunicación UCI	Requerido
i_engine_path	str	Ruta al motor de ajedrez	/opt/picochess/engines/aarch64/a-stockf
i_player_color	chess.Color	Color del jugador humano	chess.WHITE
i_fen	str	Posición inicial FEN	"" (posición inicial)
i_comment_file	str	Archivo de comentarios personalizados	""
i_lang	str	Idioma para comentarios	"en"
i_always_run_tutor			

bool | Forzar análisis continuo | False | | loop | asyncio.Loop | Bucle de eventos asíncrono | None |

Configuración de Motores

Motor Profundo (best_engine)

`python options = { "Contempt": 0, # Sin sesgo posicional "Threads": NUM_THREADS # 1-2 hilos según configuración } limit = Limit(depth=DEEP_DEPTH) # 17 niveles de profundidad multipv = VALID_ROOT_MOVES # 50 variantes principales `

Motor Obvio (obvious_engine)

`python options = { "Contempt": 0, "Threads": LOW_NUM_THREADS # 1 hilo } limit = Limit(depth=LOW_DEPTH) # 5 niveles de profundidad multipv = LOW_ROOT_MOVES # 50 variantes `

Motores de Análisis

Gestión de Motores

Apertura de Motores

`python async def open_engine(self): """Abre ambos motores de análisis de forma asíncrona""" if not self.best_engine: options = {"Contempt": 0, "Threads": c.NUM_THREADS} self.best_engine = await self._load_engine(options, "best picotutor") if not self.obvious_engine: options = {"Contempt": 0, "Threads": c.LOW_NUM_THREADS} self.obvious_engine = await self._load_engine(options, "obvious picotutor") `

Inicio de Análisis

`python async def start(self): """Inicia el análisis en ambos motores""" # Motor profundo if self.best_engine and self.best_engine.loaded_ok(): limit = Limit(depth=c.DEEP_DEPTH) # 17 ply multipv = c.VALID_ROOT_MOVES # 50 variantes await self.best_engine.start_analysis(self.board, limit=limit, multipv=multipv) # Motor obvio (con pequeño retraso) await asyncio.sleep(0.05) if self.obvious_engine and self.obvious_engine.loaded_ok(): limit = Limit(depth=c.LOW_DEPTH) # 5 ply multipv = c.LOW_ROOT_MOVES # 50 variantes await self.obvious_engine.start_analysis(self.board, limit=limit, multipv=multipv) `

Control de Análisis

Condiciones para Ejecutar el Tutor

`python def _should_run_tutor(self) -> bool: """Determina si el tutor debe ejecutarse""" if not (self.coach_on or self.watcher_on): return False # Usuario ha desactivado el tutor # Ejecutar si analizamos ambos lados O si es el turno del usuario return self.analyse_both_sides or self.board.turn == self.user_color `

Parada de Motores

`python def stop(self): """Detiene el análisis en ambos motores""" if self.best_engine: self.best_engine.stop() if self.obvious_engine: self.obvious_engine.stop() `

Sistema de Evaluación de Jugadas

Umbrales de Evaluación

Los umbrales están definidos en picotutor_constants.py :

Jugadas Malas

| Símbolo | Umbral | Descripción | |---------|--------|-------------| | ?? | 250+ centipeones | Error grave (blunder) | | ? | 150+ centipeones | Error (mistake) | | ?! | 30+ centipeones | Jugada dudosa |

Jugadas Buenas

| Símbolo | Umbral | Descripción | |---------|--------|-------------| | !! | 0 centipeones + mejora 350+ | Jugada brillante | | ! | ≤30 centipeones + mejora 250+ | Jugada buena | | !? | <30 centipeones + condiciones especiales | Jugada interesante |

Proceso de Evaluación

1. Captura de Análisis

`python async def eval_legal_moves(self, turn: chess.Color, analysed_move_already_done: bool = True): """Actualiza información de análisis desde snapshot del motor""" # Preparar tablero antes de la jugada del usuario board_before_usermove = self.board.copy() if analysed_move_already_done: board_before_usermove.pop() # Obtener análisis de ambos motores obvious_result = await self.obvious_engine.get_analysis(board_before_usermove) self.obvious_info[turn] = obvious_result.get("info") best_result = await self.best_engine.get_analysis(board_before_usermove) self.best_info[turn] = best_result.get("info") `

2. Evaluación de Jugada del Usuario

`python def get_user_move_eval(self) -> tuple: """Evalúa la jugada anterior del usuario""" # Obtener mejor jugada y puntuación best_pv, best_move, best_score, best_mate = self.best_moves[self.board.turn][0] # Obtener jugada actual del usuario current_pv, current_move, current_score, current_mate = self.best_history[self.board.turn][-1] #

```
Calcular diferencias best_deep_diff = best_score - current_score # Pérdida de centipeones # Aplicar lógica de evaluación if best_deep_diff > c.VERY_BAD_MOVE_TH: eval_string = "??" elif best_deep_diff > c.BAD_MOVE_TH: eval_string = "?" # ... más condiciones return eval_string, current_mate `
```

Algoritmo de Evaluación Detallado

Cálculo de Diferencias

`python

Diferencias principales

```
best_deep_diff = best_score - current_score # Pérdida vs mejor jugada deep_low_diff = current_score - low_score # Diferencia profundidad score_hist_diff = current_score - before_score # Cambio histórico `
```

Condiciones para Símbolos

Errores Graves (??) `python if best_deep_diff > VERY_BAD_MOVE_TH: # 250+ centipeones eval_string = "??" `

Errores (?) `python elif best_deep_diff > BAD_MOVE_TH: # 150+ centipeones eval_string = "?" `

Jugadas Dudosas (?!) `python elif (not approximations_in_use and history_in_use and best_deep_diff > DUBIOUS_TH and # >30 centipeones abs(deep_low_diff) > UNCLEAR_DIFF and # >70 diferencia score_hist_diff > POS_INCREASE): # >50 aumento eval_string = "?!" `

Jugadas Brillantes (!!) `python if (best_deep_diff <= VERY_GOOD_MOVE_TH and # ≤0 centipeones deep_low_diff > VERY_GOOD_IMPROVE_TH): # >350 mejora if not (best_score == 99999 and best_mate == current_mate and legal_no <= 2): eval_string = "!!" `

Análisis de Aperturas

Sistema de Identificación de Aperturas

PicoTutor incluye un sistema completo de identificación de aperturas basado en:

1. Base de Datos ECO

- **Archivo:** chess-eco_pos.txt
- **Formato:** CSV con delimitador |
- **Campos:** eco , opening_name , moves

2. Base de Datos FEN

- **Archivo:** opening_name_fen.txt
- **Formato:** Líneas alternadas FEN/Nombre
- **Uso:** Identificación por posición

Proceso de Identificación

Búsqueda por Secuencia de Jugadas

`python def _find_longest_matching_opening(self, played: str) -> Tuple[str, str, str]: """Encuentra la apertura más larga que coincida con las jugadas""" opening_name = moves = eco = "" for opening in self.book_data: if played[:len(opening.get("moves"))] == opening.get("moves"): if len(opening.get("moves")) > len(moves): opening_name = opening.get("opening_name") moves = opening.get("moves") eco = opening.get("eco") return opening_name, moves, eco `

Búsqueda por Posición FEN

`python def get_fen_opening(self): """Identifica apertura por posición FEN actual""" fen = self.board.board_fen() for i, line in enumerate(self.book_fen_data): line_list = line.split() if line_list[0] == fen: opening_name = self.book_fen_data[i + 1] return opening_name.strip(), True return "", False `

Información de Apertura Disponible

Método Principal

`python def get_opening(self) -> Tuple[str, str, str, bool]: """ Retorna información completa de la apertura actual Returns: eco: Código ECO (ej: "E20") opening_name: Nombre de la apertura (ej: "Nimzo-Indian Defense") moves: Secuencia de jugadas (ej: "1.d4 Nf6 2.c4 e6 3.Nc3 Bb4") inside_book_opening: True si aún estamos en teoría de apertura """ `

Gestión de Partidas PGN

Carga y Navegación de Partidas

PicoTutor permite cargar partidas PGN y navegar por ellas paso a paso:

Carga de Partida PGN

`python def set_pgn_game_to_step(self, pgn_game: chess.pgn.Game): """Almacena una partida PGN cargada para navegación paso a paso""" self.pgn_game = pgn_game `

Navegación por Jugadas

`python def get_next_pgn_move(self, current_board: chess.Board) -> chess.Move: """ Obtiene la siguiente jugada en la partida PGN cargada Args: current_board: Tablero actual del juego Returns: chess.Move: Siguiente jugada o None si no hay más jugadas """ if not self.pgn_game: return None node = self.pgn_game # Caso especial: posición inicial if current_board.fen() == chess.Board.starting_fen: if node.variations: return node.variations[0].move return None # Buscar posición actual en el árbol PGN while node.variations: next_node = node.variations[0] temp_board = node.board() temp_board.push(next_node.move) if temp_board.fen() == current_board.fen(): # Posición encontrada, retornar siguiente jugada if next_node.variations: return next_node.variations[0].move return None node = next_node return None `

Uso Práctico de PGN

Análisis de Partidas Maestras

1. **Cargar PGN**: Usar el endpoint /upload en la interfaz web 2. **Navegar**: Usar botón de pausa para avanzar jugada por jugada 3. **Analizar**: PicoTutor evalúa cada jugada automáticamente 4. **Guardar**: Las evaluaciones se guardan en /opt/picochess/games/

Flujo de Trabajo Típico

` 1. Usuario carga archivo PGN → set_pgn_game_to_step() 2. PicoChess llama → get_next_pgn_move() para cada posición 3. Usuario presiona pausa → Avanza a siguiente jugada 4. PicoTutor evalúa → Genera comentarios y evaluaciones 5. Al final → Guardar partida con análisis

completo `

Modos de Funcionamiento

Estados del Tutor

PicoTutor opera en diferentes modos según la configuración:

1. Modo Watcher (Observador)

`python self.watcher_on = True # Análisis activo pero sin coaching self.coach_on = False # Sin sugerencias activas `

- **Función:** Análisis silencioso en segundo plano
- **Uso:** Recopilación de estadísticas sin interferir
- **Evaluación:** Genera evaluaciones pero no las muestra

2. Modo Coach (Entrenador)

`python self.watcher_on = True # Análisis activo self.coach_on = True # Coaching activo `

- **Función:** Análisis completo con sugerencias
- **Uso:** Entrenamiento activo del usuario
- **Evaluación:** Muestra evaluaciones y sugerencias

3. Modo Explorer (Explorador)

`python self.explorer_on = True # Exploración de aperturas activa `

- **Función:** Análisis especializado en aperturas
- **Uso:** Aprendizaje de teoría de aperturas
- **Evaluación:** Enfoque en jugadas de libro

4. Modo Comments (Comentarios)

`python self.comments_on = True # Comentarios contextuales activos `

- **Función:** Generación de comentarios automáticos

- **Uso**: Partidas comentadas automáticamente
- **Evaluación**: Comentarios basados en evaluación

Configuración de Modos

Método Principal

`python async def set_status(self, watcher=False, coach=PicoCoach.COACH_OFF, explorer=False, comments=False): """Configura el estado operativo del tutor""" # Convertir coach enum a boolean b_coach = coach != PicoCoach.COACH_OFF # Actualizar estados self.watcher_on = watcher self.coach_on = b_coach self.explorer_on = explorer self.comments_on = comments # Iniciar o detener análisis según sea necesario await self._start_or_stop_as_needed() `

Control Automático

`python async def _start_or_stop_as_needed(self): """Inicia o detiene el análisis según la configuración actual""" if self._should_run_tutor(): await self.start() # Análisis normal (profundo + obvio) elif self.always_run_tutor: await self.start() # Forzar análisis continuo else: self.stop() # Detener análisis `

Análisis Bilateral

Configuración

`python async def set_mode(self, analyse_both_sides: bool, deep_limit_depth: int = None): """ Configura el modo de análisis Args: analyse_both_sides: Si True, analiza jugadas de ambos bandos deep_limit_depth: Profundidad personalizada (None = usar default 17) """ self.deep_limit_depth = deep_limit_depth if self.analyse_both_sides != analyse_both_sides: self.analyse_both_sides = analyse_both_sides await self._start_or_stop_as_needed() `

Casos de Uso

- **analyse_both_sides = False**: Solo analiza jugadas del usuario humano

- **analyse_both_sides = True**: Analiza jugadas de ambos bandos (útil para análisis de PGN)

Sistema de Comentarios

Tipos de Comentarios

PicoTutor maneja dos tipos de comentarios:

1. Comentarios Específicos del Motor

- **Archivo**: Definido por usuario en inicialización
- **Uso**: Comentarios específicos para un motor particular
- **Formato**: Un comentario por línea

2. Comentarios Generales

- **Archivo**: /opt/picochess/engines/{arch}/general_game_comments_{lang}.txt

- **Uso**: Comentarios generales del juego
- **Idiomas**: Soporta múltiples idiomas (en, es, de, etc.)

Configuración de Comentarios

Inicialización

`python def _setup_comments(self, i_lang, i_comment_file): """Configura el sistema de comentarios""" # Comentarios específicos del motor if i_comment_file: try: with open(i_comment_file) as fp: self.comments = fp.readlines() self.comment_no = len(self.comments) except OSError: self.comments = [] # Comentarios generales try: arch = platform.machine() general_file = f"/opt/picochess/engines/{arch}/general_game_comments_{i_lang}.txt" with open(general_file) as fp: self.comments_all = fp.readlines() self.comment_all_no = len(self.comments_all) except (OSError, IOError):

self.comments_all = [] `

Generación de Comentarios

Método Principal

`python def get_game_comment(self, pico_comment=PicoComment.COM_OFF, com_factor=0): """ Genera un comentario aleatorio basado en la configuración Args: pico_comment: Tipo de comentario (COM_OFF, COM_ON_ENG, COM_ON_ALL) com_factor: Factor de frecuencia (0-100, donde 100 = siempre) Returns: str: Comentario seleccionado o cadena vacía """ if com_factor == 0: return "" # Calcular probabilidad range_fac = round(100 / com_factor) max_range = self.comment_no * range_fac max_range_all = self.comment_all_no * range_fac if pico_comment == PicoComment.COM_ON_ENG: # Solo comentarios específicos del motor if self.comments and self.comment_no > 0: index = randint(0, max_range) if index <= self.comment_no - 1: return self.comments[index].strip() elif pico_comment == PicoComment.COM_ON_ALL: # Primero intentar comentarios específicos, luego generales if self.comments and self.comment_no > 0: index = randint(0, max_range) if index <= self.comment_no - 1: return self.comments[index].strip() # Fallback a comentarios generales if self.comments_all and self.comment_all_no > 0: index = randint(0, max_range_all) if index <= self.comment_all_no - 1: return self.comments_all[index].strip() return "" `

Integración con Evaluaciones

Los comentarios se integran automáticamente con las evaluaciones de jugadas:

`python

En get_user_move_eval(), los comentarios se almacenan junto con las evaluaciones

e_value = { "nag": PicoTutor.symbol_to_nag(eval_string), "best_move": board_before_usermove.san(best_move), "user_move": board_before_usermove.san(current_move), "CPL": best_deep_diff, "score": current_score } self.evaluated_moves[e_key] = e_value `

API y Métodos Principales

Métodos de Control del Ciclo de Vida

Inicialización y Configuración

`python async def open_engine(self) """Abre los motores de análisis"""

async def set_mode(self, analyse_both_sides: bool, deep_limit_depth: int = None) """Configura el modo de análisis"""

async def set_status(self, watcher=False, coach=PicoCoach.COACH_OFF, explorer=False, comments=False) """Establece el estado operativo del tutor"""

async def set_user_color(self, i_user_color, analyse_both_sides: bool) """Configura el color del usuario y modo de análisis""" `

Gestión de Posiciones

`python async def set_position(self, game: chess.Board, new_game: bool = False) """Sincroniza la posición del tablero con el juego principal"""

```
async def push_move(self, i_uci_move: chess.Move, game: chess.Board) -> bool """Informa de una jugada realizada y la evalúa"""

async def pop_last_move(self, game: chess.Board) -> bool """Maneja el retroceso de jugadas (takeback)"""


def newgame(self) """Reinicia todo para una nueva partida""" `
```

Métodos de Análisis

Evaluación de Jugadas

```
`python async def eval_legal_moves(self, turn: chess.Color, analysed_move_already_done: bool = True) """Actualiza información de análisis desde snapshot del motor"""

def eval_user_move(self, user_move: chess.Move) """Evalúa y almacena la jugada del usuario en el historial"""

def get_user_move_eval(self) -> tuple """Obtiene la evaluación de la jugada anterior del usuario"""


async def get_pos_analysis(self) """Obtiene análisis completo de la posición actual""" `
```

Información de Análisis

```
`python async def get_analysis(self) -> dict """Obtiene información de análisis del motor principal"""

def get_user_move_info(self) -> tuple """Retorna (hint_move, pv) de la jugada del usuario"""
```

```
def can_use_coach_analyser(self) -> bool """Verifica si el analizador está activo y disponible""" `
```

Métodos de Información

Aperturas

```
`python def get_opening(self) -> Tuple[str, str, str, bool] """Retorna información completa de la apertura actual"""
```

```
def get_fen_opening(self) """Identifica apertura por posición FEN""" `
```

Estado del Juego

```
`python def is_same_board(self, game: chess.Board) -> bool """Verifica si el tablero del tutor está sincronizado"""

def get_user_color(self) """Retorna el color del usuario"""

def get_eng_long_name(self) """Retorna el nombre completo del motor utilizado"""
```

```
def get_eval_moves(self) -> dict """Retorna diccionario de todas las jugadas evaluadas""" `
```

Métodos de PGN

Gestión de Partidas PGN

```
`python def set_pgn_game_to_step(self, pgn_game: chess.pgn.Game) """Almacena una partida PGN para navegación paso a paso"""
```

```
def get_next_pgn_move(self, current_board: chess.Board) -> chess.Move
"""Obtiene la siguiente jugada en la partida PGN cargada""" `
```

Métodos de Utilidad

Conversión de Formatos

```
`python @staticmethod def symbol_to_nag(eval_string: str) -> int
"""Convierte símbolo de evaluación (!) a formato NAG"""

@staticmethod def nag_to_symbol(nag: int) -> str """Convierte formato
NAG a símbolo de evaluación"""


@staticmethod def get_score(info: InfoDict, turn: chess.Color =
chess.WHITE) -> tuple """Extrae (move, score, mate) de InfoDict""" `
```

Manejo de Jugadas y Turnos

```
`python @staticmethod def halfmove_to_fullmove(halfmove_nr: int,
known_turn: chess.Color = None) -> Tuple """Convierte número de media
jugada a jugada completa"""


@staticmethod def printable_move_filler(halfmove_nr: int, turn:
chess.Color) -> str """Retorna string de relleno para mostrar jugadas
(ej: '1. ' o '1... ')""" `
```

Configuración Avanzada

Parámetros de Rendimiento

Configuración de Hilos

`python

En picotutor_constants.py

NUM_THREADS = 1 # Hilos para motor profundo (1-2 recomendado)
LOW_NUM_THREADS = 1 # Hilos para motor obvio (siempre 1) `

Profundidad de Análisis

`python DEEP_DEPTH = 17 # Profundidad motor principal (17 ply)
LOW_DEPTH = 5 # Profundidad motor obvio (5 ply) `

Número de Variantes

`python VALID_ROOT_MOVES = 50 # Variantes principales analizadas
LOW_ROOT_MOVES = 50 # Variantes obvias analizadas `

Umbrales de Evaluación Personalizables

Errores

`python VERY_BAD_MOVE_TH = 250 # Umbral para ?? (blunder) BAD_MOVE_TH = 150 # Umbral para ? (mistake) DUBIOUS_TH = 30 # Umbral para ?! (dubious) INACCURACY_TH = 20 # Umbral para imprecisiones `

Jugadas Buenas

`python VERY_GOOD_MOVE_TH = 0 # Umbral para !! (brilliant)
GOOD_MOVE_TH = 30 # Umbral para ! (good) INTERESTING_TH = 30 # Umbral para !? (interesting) `

Diferencias Contextuales

`python VERY_GOOD_IMPROVE_TH = 350 # Mejora requerida para !! GOOD_IMPROVE_TH = 250 # Mejora requerida para ! UNCLEAR_DIFF = 70 # Diferencia para posiciones poco claras POS_INCREASE = 50 # Aumento posicional para ?! POS_DECREASE = -50 # Disminución posicional para !? `

Optimización para Diferentes Hardware

Raspberry Pi 4/5

`python NUM_THREADS = 2 # Aprovechar múltiples cores DEEP_DEPTH = 17 # Profundidad completa VALID_ROOT_MOVES = 50 # Análisis completo `

Raspberry Pi 3 o Hardware Limitado

`python NUM_THREADS = 1 # Un solo hilo DEEP_DEPTH = 15 # Reducir profundidad VALID_ROOT_MOVES = 30 # Menos variantes `

Computadoras de Escritorio

`python NUM_THREADS = 2 # O más según CPU DEEP_DEPTH = 20 # Mayor profundidad VALID_ROOT_MOVES = 100 # Más variantes `

Configuración de Idiomas

Archivos de Comentarios por Idioma

` /opt/picochess/engines/{arch}/general_game_comments_en.txt # Inglés /opt/picochess/engines/{arch}/general_game_comments_es.txt # Español /opt/picochess/engines/{arch}/general_game_comments_de.txt # Alemán /opt/picochess/engines/{arch}/general_game_comments_fr.txt # Francés `

Configuración en picochess.ini

`ini [tutor] language = es # Idioma para comentarios comment-factor = 50 # Frecuencia de comentarios (0-100) coach-on = true # Activar modo coach watcher-on = true # Activar modo watcher `

Resolución de Problemas

Problemas Comunes y Soluciones

1. Motor No Carga

Síntomas: best engine loading failed in Picotutor

Causas Posibles:

- Ruta del motor incorrecta
- Motor no ejecutable
- Falta de permisos

Soluciones: `bash

Verificar ruta del motor

```
ls -la /opt/picochess/engines/aarch64/a-stockf
```

Dar permisos de ejecución

```
chmod +x /opt/picochess/engines/aarch64/a-stockf
```

Verificar arquitectura

```
uname -m # Debe coincidir con carpeta (aarch64 o x86_64) `
```

2. Tableros Desincronizados

Síntomas: picotutor board not in sync

Causas:

- Jugadas ilegales
- Errores en takeback
- Cambio de posición sin notificar

Soluciones: `python

Forzar resincronización

```
await picotutor.set_position(game_board, new_game=False)
```

Verificar sincronización

```
if not picotutor.is_same_board(game_board): logger.warning("Boards out of sync, resetting position") await picotutor.set_position(game_board)
`
```

3. Análisis No Funciona

Síntomas: No hay evaluaciones de jugadas

Verificaciones: `python

Verificar estado del tutor

```
if not picotutor.can_use_coach_analyser(): logger.debug("Coach analyser not available")
```

Verificar configuración

```
logger.debug(f"Watcher: {picotutor.watcher_on}") logger.debug(f"Coach: {picotutor.coach_on}") logger.debug(f"User color: {picotutor.user_color}") `
```

4. Rendimiento Lento

Síntomas: Análisis muy lento o sistema sobrecargado

Optimizaciones: `python

Reducir profundidad

```
picotutor.deep_limit_depth = 12 # En lugar de 17
```

Reducir variantes

Modificar en picotutor_constants.py

```
VALID_ROOT_MOVES = 20 # En lugar de 50
```

Usar un solo hilo

```
NUM_THREADS = 1 `
```

Debugging y Logging

Activar Logging Detallado

```
`python import logging
logging.getLogger('picotutor').setLevel(logging.DEBUG)
```

Métodos de debug disponibles

```
picotutor.log_sync_info() # Estado de sincronización
picotutor.log_pv_lists() # Listas de variantes principales
picotutor.log_eval_moves() # Jugadas evaluadas `
```

Verificar Estado Interno

`python

Verificar motores

```
print(f"Best engine loaded: {picotutor.best_engine.loaded_ok()}")
print(f"Obvious engine loaded: {picotutor.obvious_engine.loaded_ok()}")
```

Verificar análisis

```
print(f"Analysis running:
{picotutor.best_engine.is_analyser_running()}")
```

Verificar historial

```
print(f"Best history length:
{len(picotutor.best_history[chess.WHITE])}") print(f"Obvious history
length: {len(picotutor.obvious_history[chess.WHITE])}") `
```

Archivos de Log Importantes

Ubicaciones de Logs

```
` /opt/picochess/logs/picochess.log # Log principal
/opt/picochess/logs/engine.log # Log de motores UCI `
```

Información Útil en Logs

- Errores de carga de motores
- Problemas de sincronización
- Evaluaciones de jugadas
- Errores de análisis

Recuperación de Errores

Reinicio Suave

```
`python
```

Detener análisis

```
picotutor.stop()
```

Reiniciar con posición actual

```
await picotutor.set_position(current_board) await picotutor.start() `
```

Reinicio Completo

```python
```

Cerrar motores

```
await picotutor.exit_or_reboot_cleanups()
```

Crear nueva instancia

```
picotutor = PicoTutor(ucishell, engine_path, user_color, fen,
comment_file, lang, always_run_tutor, loop) await
picotutor.open_engine() ``
```

Conclusión

PicoTutor es un sistema complejo y potente que proporciona análisis de ajedrez en tiempo real con múltiples funcionalidades:

Características Principales Resumidas:

- **Análisis dual**: Motor profundo (17 ply) y obvio (5 ply)
- **Evaluación automática**: Símbolos estándar de ajedrez (!, ?, !!, ??, !?, ?!)
- **Identificación de aperturas**: Base de datos ECO completa
- **Soporte PGN**: Navegación y análisis paso a paso
- **Múltiples modos**: Watcher, Coach, Explorer, Comments
- **Sistema de comentarios**: Multiidioma y contextual
- **API completa**: Métodos para integración con PicoChess

Casos de Uso Típicos:

1. **Entrenamiento**: Análisis en tiempo real de partidas 2. **Estudio**: Análisis de partidas maestras desde PGN 3. **Análisis post-partida**: Revisión detallada con evaluaciones 4. **Aprendizaje de aperturas**: Identificación y exploración de teoría

Configuración Recomendada:

- **Principiantes**: Coach ON, Watcher ON, Comments ON
- **Jugadores intermedios**: Watcher ON, Comments selectivos
- **Análisis de PGN**: analyse_both_sides=True, profundidad máxima
- **Hardware limitado**: Reducir profundidad y variantes

PicoTutor representa una herramienta educativa poderosa que combina la potencia de los motores de ajedrez modernos con una interfaz intuitiva y funcionalidades pedagógicas avanzadas.

Este manual cubre la funcionalidad completa de PicoTutor v4.x. Para actualizaciones y nuevas características, consulte el repositorio oficial de PicoChess.