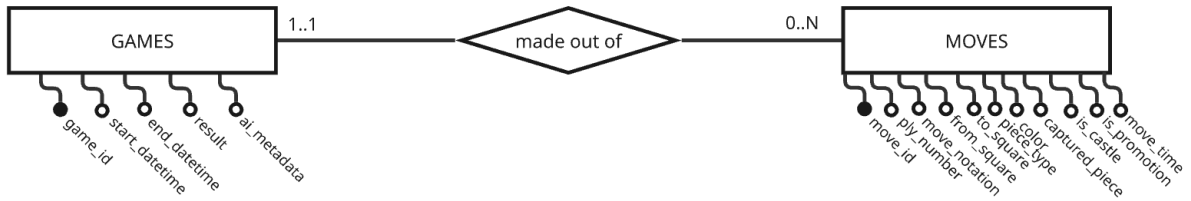
CHESS AI

BRIEF PROJECT OVERVIEW

This project builds an **object-oriented chess engine** with a **PyGame interface**, blending move logic with visual interaction. The core architecture starts with a **father class: piece** (handling color, position, status, and value), extended via subclasses, each subclass implements the movement rules of that specific piece. The **8x8 board matrix** manages piece states, validates moves, and supports debug rendering.

DATABASE STRUCTURE



CHOICES

Designing a chess AI database requires a structured foundation for learning and analysis. Each field must serve a functional role in move evaluation, with this data, a fitness function will be made, the fitness function makes it possible to give a score to the moves the AI does during a game, understanding what behavior is more likely to bring a victory.

Ply numbering provides a game timeline, essential for move ordering, reconstructing positions, and aligning moves with outcomes in learning models.

Captured pieces track material flow, aiding in evaluating exchanges, sacrifices, and reward signals for training.

Special move flags (castling, promotion) mark critical moments, helping AI detect phase transitions and optimize search behavior.

Move timing offers behavioral insights, supporting adaptive search strategies and performance profiling.

JSON metadata stores algorithm-specific data (search stats, evaluation breakdowns), enabling flexible experimentation.

Full move history allows position reconstruction, branching analysis, and trajectory-based training.

Color data ensures correct perspective in evaluations, symmetry-aware training, and proper move generation.

Standard algebraic notation ensures compatibility with engines, GUIs, and analytical tools.

Data integrity (foreign keys, NOT NULL constraints, strict types) maintains consistency, automates cleanup, and prevents malformed records.

This schema supports **training, analysis, and interoperability**, enabling scalable AI development in a structured domain like chess.

RELATIONSHIP SCHEMA

games(**game_id**, start_datetime, end_datetime, duration_seconds, result, ai_metadata)
moves(**move_id**, *game_id*, ply_number, move_notation, from_square, to_square, piece_type, color, captured_piece, is_castle, is_promotion, move_time)

ATTRIBUTES MEANING

Games Table

- **game_id**: unique identifier (auto-incremented)
- **start_datetime/end_datetime**: game duration
- **duration_seconds**: pre-calculated for faster analytics
- **result**: standard chess results, '*' for unfinished
- **ai_metadata**: flexible JSON field for evolutionary algorithm data

Moves Table

- **move_id**: unique identifier (auto-incremented)
- **game_id**: originating game's foreign key
- **ply_number**: half-move counter (essential for move ordering)
- **move_notation**: standard algebraic notation
- **from_square/to_square**: chess movement (coordinates)
- **piece_type**: piece codes
- **color**: player color making the move
- **captured_piece**: records captured piece
- **is_castle/is_promotion**: flags for special moves
- **move_time**: timestamp of the move

CREATION QUERIES

GAMES

```
CREATE TABLE games (
   game_id INT AUTO_INCREMENT PRIMARY KEY,
   start_datetime DATETIME NOT NULL,
   end_datetime DATETIME NULL,
   duration_seconds INT NULL,
   result ENUM('1-0', '0-1', '1/2-1/2', '*') NULL,
   ai_metadata JSON NULL,
   INDEX (start_datetime)
) ;
```

MOVES

```
CREATE TABLE moves (
    move_id INT AUTO_INCREMENT PRIMARY KEY,
    game_id INT NOT NULL,
    ply_number INT NOT NULL,
    move_notation VARCHAR(10) NOT NULL,
    from_square CHAR(2) NOT NULL COMMENT 'a1-h8',
    to_square CHAR(2) NOT NULL COMMENT 'a1-h8',
    piece_type CHAR(1) NOT NULL COMMENT 'P,N,B,R,Q,K',
    color CHAR(5) NOT NULL COMMENT 'white/black',
    captured_piece CHAR(1) NULL,
    is_castle BOOLEAN DEFAULT FALSE,
    is_promotion BOOLEAN DEFAULT FALSE,
    move_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (game_id) REFERENCES games(game_id) ON DELETE CASCADE,
    INDEX (game_id, ply_number)
) ;
```