Chess Bitboard Engine

Overview

A chess engine built from scratch using **bitboards** for efficient board representation and move generation. The engine supports all standard chess rules, advanced mechanics (castling, en passant, promotion), and includes a **minimax search with alpha-beta pruning**. It can be played interactively in the terminal (human vs human or human vs engine) and has been validated using **perft testing**.

Features

- Full chess rules implemented
- Efficient move generation with **magic bitboards**
- Minimax + alpha-beta pruning search engine
- Evaluation with material, piece-square tables, development, pawn pushes, and safety
- Supports **FEN input/output**
- Perft validation up to depth `n`
- Playable via terminal (PvP and PvE modes)

Project Structure

```
├── bitboard.py         # Core bitboard constants, masks, and board state
├── board.py            # Board class for move execution and undo
├── moves.py            # Piece move and attack generation
├── move_generate.py    # Generates all moves for active side
├── legal_moves.py      # Filters legal moves (king safety)
├── castling.py         # Castling rights, execution, undo
├── en_passant.py       # En passant detection, execution, undo
├── promotion.py        # Pawn promotion logic
├── move_record.py      # Move and evaluation history
├── fen.py              # FEN parsing and board setup
├── evaluations.py      # Position evaluation heuristics
├── evaluation_tables.py# Piece-square evaluation tables
├── search_engine.py    # Minimax with alpha-beta pruning
├── perft.py            # Perft testing and statistics
├── utils.py            # Helper functions (coordinates, counters, debugging)
├── threats.py          # Threat maps (attacked squares)
├── main_terminal.py    # Human vs human game loop
├── player_vs_engine.py # Human vs engine game loop
└── _init_.py           # Initializes masks, magic tables, and attack sets
```

Installation & Usage

Prerequisites

- Python 3.9+

Run Perft Test

```
python main_terminal.py
```

The program will ask if you want to load a FEN. After initialization, it runs perft testing by default.

Play Against Engine

```
python player_vs_engine.py
```

Choose your side and play using standard algebraic coordinates (e.g., `e2` → `e4`).

Play PvP in Terminal

```
python main_terminal.py
```

Choose sides and enter moves manually.

Example

```
  a b c d e f g h
 +-----------------+
8 | r n b q k b n r | 8
7 | p p p p p p p p | 7
6 | . . . . . . . . | 6
5 | . . . . . . . . | 5
4 | . . . . . . . . | 4
3 | . . . . . . . . | 3
2 | P P P P P P P P | 2
1 | R N B Q K B N R | 1
 +-----------------+
  a b c d e f g h
```

Next Steps (Optional Improvements)

- Add **transposition tables** (Zobrist hashing)
- Implement **quiescence search** and **move ordering heuristics**
- Expand evaluation (king safety, mobility, pawn structure)
- Build a **GUI** with `pygame` or `tkinter`
- Export PGN/FEN for external GUIs (Arena, Lichess bots)

License

This project is for educational and portfolio purposes. Free to use and extend.