chess

Chess bot written in rust

Improvement Roadmap

Speed Optimizations

- [x] **Incremental PST Evaluation** ✓ DONE

  ~~Currently the evaluation function iterates all pieces on every leaf node, which is O(n_pieces) per eval call.~~ Now maintains running PST scores that update incrementally during `make_move` / `unmake_move`. Evaluation is now O(1) per position.

- [x] **Remove `pieces: Vec<Piece>` Redundancy** ✓ PARTIAL

  Added `iter_pieces()` method using bitboards. Updated hot paths (evaluation, material counting, pin detection) to use bitboards. The pieces Vec is still used in make/unmake_move but is no longer on the critical evaluation path.

- [ ] **Lazy/Incremental Attack Map Updates** ⚠️ ATTEMPTED

  `recompute_attack_maps()` is called on every `make_move` and `unmake_move`, regenerating all sliding piece attacks from scratch. Lazy evaluation was attempted using atomics for thread-safety but the atomic overhead (~30-40% perft slowdown) outweighed the benefits. Future approaches: incremental updates that only recalculate affected sliding piece rays, or magic bitboards which would make full recomputation fast enough.

- [ ] **Stack-Allocated Move Lists**

  `get_legal_moves()` allocates a new `Vec<Move>` per node, causing heap allocation pressure. Replace with `ArrayVec<Move, 256>` (stack-allocated) or a thread-local move list pool. The maximum legal moves in any chess position is 218, so a fixed-size array suffices.

- [ ] **Magic Bitboards for Sliding Pieces**

  Currently using classical ray attacks with blocker detection via `trailing_zeros` / `leading_zeros`. Magic bitboards use a precomputed lookup table indexed by `(occupancy * magic) >> shift`, reducing sliding piece attack generation to a single array lookup. Typically provides ~2x speedup for move generation.

Strength Optimizations

- [x] **Enable Endgame PST Tables** ✓ DONE

~~The codebase defines `PAWNS_END` and `KING_END` tables but `is_endgame` is hardcoded to `false`.~~ Now uses tapered evaluation with phase detection: `phase = (queens*4 + rooks*2 + bishops + knights)`. Interpolates between middlegame and endgame PST scores based on remaining material.

- [x] **Killer Move Heuristic** ✓ DONE

  ~~Track the last 2 quiet moves that caused a beta cutoff at each ply depth.~~ Implemented in `SearchState` struct. Stores 2 killer moves per ply, prioritized in move ordering after TT move and captures (90,000 for first killer, 80,000 for second).

- [x] **History Heuristic** ✓ DONE

  ~~Maintain a `[Color][From][To]` table that accumulates a bonus each time a quiet move causes a beta cutoff, scaled by depth².~~ Implemented in `SearchState`. History scores indexed by `[color][from_sq][to_sq]`, used for quiet move ordering. Scores aged at each depth iteration to gradually forget old information.

- [x] **Principal Variation Search (PVS)** ✓ DONE

  ~~After searching the first move with a full window, search remaining moves with a null window `(-alpha-1, -alpha)`. If the null window search fails high, re-search with the full window.~~ Implemented in `negamax_with_tt_mut`. First move uses full window, subsequent moves use null window (-alpha-1, -alpha), with full re-search on fail high. Combined with LMR for later quiet moves. Benchmarks show ~12-28% faster search at depths 5-6.

- [x] **Aspiration Windows** ✓ DONE

  ~~In iterative deepening, use a narrow window centered on the previous iteration's score (e.g., ±25 centipawns) instead of (-∞, +∞).~~ Implemented in `iterative_deepening` and `iterative_deepening_timed`. After depth 1, uses ±25cp window centered on previous score. On fail-low/fail-high, doubles window and re-searches. Falls back to full window if window exceeds ±200cp.

- [x] **Check Extensions** ✓ DONE

  ~~When a move gives check, extend the search depth by 1 ply.~~ Implemented in `negamax_with_tt_mut`. After making a move, check if opponent is in check; if so, extend search depth by 1. Also prevents LMR from being applied to moves that give check. This avoids horizon effects where forcing check sequences are cut off prematurely.

- [x] **Static Exchange Evaluation (SEE)** ✓ DONE

  ~~Before searching a capture in quiescence, simulate the full exchange sequence on that square to determine if the capture is winning, equal, or losing.~~ Implemented `see()` function that simulates exchange sequences. In quiescence search, prunes losing captures when attacker_value > victim_value. Uses least-valuable-attacker ordering for accurate exchange simulation.

- [x] **Futility Pruning** ✓ DONE

  ~~At low depths (1-2 ply from leaf), if the static evaluation plus a margin is still below alpha, skip searching quiet moves entirely.~~ Implemented in `negamax_with_tt_mut`. At depths 1-2, skips

quiet moves (non-captures, non-promotions) if static_eval + margin < alpha. Margins: depth 1 = 200cp, depth 2 = 500cp. Not applied when in check or for the first move.

- [ ] **Improved Evaluation Terms**

  The current eval is material + PST only. Add:

  - **Pawn structure**: Penalize doubled pawns (-10), isolated pawns (-20), bonus for passed pawns (+20 to +100 by rank)
  - **Mobility**: Count pseudo-legal moves per piece, bonus for more active pieces
  - **King safety**: Bonus for pawn shield in front of castled king, penalty for open files near king
  - **Bishop pair**: +30 bonus for having both bishops

Implementation Priority

Phase	Task	Type	Expected Impact	Status
1	Enable endgame PST interpolation	Strength	Quick win, minimal code	✓ Done
2	Incremental PST evaluation	Speed	15-20% faster	✓ Done
3	Remove `pieces` Vec	Speed	10-15% faster	✓ Partial
4	Killer moves + history heuristic	Strength	Much better move ordering	✓ Done
5	Lazy/incremental attack maps	Speed	10-20% faster	⚠️ Attempted
6	PVS search	Strength	Better node efficiency	✓ Done
7	Check extensions	Strength	Avoids horizon effects	✓ Done
8	Magic bitboards	Speed	~2x faster movegen	
9	SEE pruning	Strength	Smaller quiescence tree	✓ Done
10	Futility pruning	Strength	Smaller search tree	✓ Done
11	Improved eval terms	Strength	Better positional play	