Cheese Engine

A Chess Engine with NNUE Evaluation

Chess: A Zero-Sum Game

What is Zero-Sum?

One player's gain = other's loss.
White wins (+1), Black loses (-1). Total: zero.

Perfect Information

Both players see entire board.
No hidden cards, no dice, no luck.

Why Can't We Solve It?

- Legal positions: 10^{44}
- Possible games: 10^{120} (Shannon Number)

Compare: Checkers solved with 5×10^{20} positions (2007)

We need smarter ways to explore the game tree

Exploring the Game Tree

We can't explore every position, so we need:

Search Algorithm

How to traverse the tree efficiently

Evaluation Function

How to score positions we reach

Negamax

Deterministic depth-first search with pruning

MCTS

Probabilistic sampling with statistics

Negamax Search

Core Insight

In a zero-sum game:

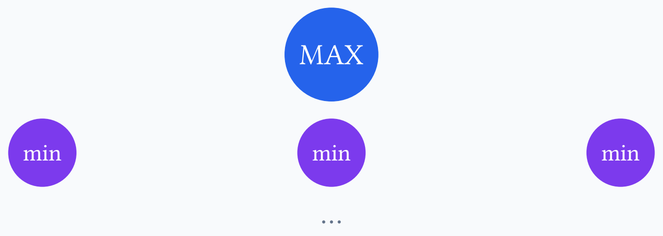
max(a, b) = −min(−a, −b)

Your best move = opponent's worst outcome. Just negate scores!

Alpha-Beta Pruning

Track best scores for both players. Skip branches that can't affect result.
Reduces $O\left(b^d\right)$ to $O\left(b^{\frac{d}{2}}\right)$ in best case!

Game Tree



Monte Carlo Tree Search (MCTS)

1. Select

UCT formula balances explore vs exploit

$$\text{UCT} = \frac{Q}{N} + c\sqrt{\ln \frac{N_p}{N}}$$

2. Expand

Add child node for unexplored move

3. Evaluate

Score position with eval function

4. Backprop

Update stats back to root

Pros

- Works well with neural network eval
- Naturally handles uncertainty
- Used by AlphaZero

Cons

- Slower in tactical positions
- Needs many iterations
- Memory intensive

Why Negamax for Chess is Hard

Branching factor: 35 moves/position At depth 6: 35^6 = 1.8 billion nodes!

We need heuristics to prune effectively:

Iterative Deepening

Search depth 1, then 2, then 3… Use shallow results to order deeper search.

Move Ordering

Try best first: TT move, captures, killers, history heuristic.

Transposition Tables

Cache positions by hash. Same position via different moves? Reuse!

Quiescence Search

Don’t stop in “noisy” positions. Extend for captures until quiet.

Evaluation: Simple Approaches

Material Counting

Sum piece values:

P=100 N=320 B=330 R=500 Q=900

Simple but misses positional nuance

Piece-Square Tables

Position-based bonuses:

- Knights love the center
- Rooks love open files
- King hides early, advances late

Separate midgame & endgame tables

Extra heuristics: Passed pawns (+), Doubled pawns (-), Bishop pair (+), King safety...

NNUE: Neural Networks for Chess

What Makes NNUE Special?

NNUE = Efficiently Updatable Neural Network Originally from Shogi (Yu Nasu, 2018)

Standard Neural Network

Every evaluation requires:

- Full forward pass
- All weights × all inputs
- 789 × hidden_size multiplications
- **Expensive!**

NNUE Approach

Exploit incremental changes:

- Most inputs unchanged between moves
- Update only what changed
- **Massive speedup!**

Key insight: A move only changes 2-4 pieces on the board

NNUE Architecture vs Standard MLP

Input Encoding (789 features)

- **768** Piece-square (12 pieces × 64 squares)
- **4** Castling rights
- **16** En passant squares
- **1** Side to move

The Sparsity Trick

768 features, but only 32 pieces! **95%+ are zero**

Network Structure

Input: 789 (sparse)

↓

Hidden layers

↓

Output: 1

(position score)

NNUE: Why CPU Beats GPU

Accumulator Technique

Maintain running sum of active features:

```
acc = W[piece1] + W[piece2] + ...
```

On each move:

- Subtract: removed piece contribution
- Add: new position contribution

No full forward pass needed!

Why Not GPU?

- Batch size = 1 (single position)
- GPU excels at large batches
- Memory transfer overhead
- CPU cache faster for small ops

SIMD Acceleration

- AVX2/AVX-512 parallel ops
- 8-16 values at once
- Perfect for accumulator

Result: Millions of evaluations per second on CPU

Implementation in Cheese Engine

Search: Negamax

- Alpha-beta with PVS
- Iterative deepening (depth 1-4)
- 16M entry transposition table
- Aspiration windows

Move Ordering: TT → Captures → Killers → History

Tech Stack

Rust, `chess` crate, `ort` (ONNX Runtime), UCI protocol

Evaluation Options

1. **Material counting**

- Simple piece sum

2. **Piece-Square Tables**

- Phase-interpolated

3. **NNUE**

- 789-dim neural network

Also: MCTS

Alternative search with UCT, 4000 iterations/move

Conclusion & Future Work

Chess AI = Search + Evaluation

NNUE bridges classical heuristics with neural networks,
enabling strong play without expensive hardware.

Key Takeaways

- Chess too complex to solve completely
- Smart heuristics make search tractable
- Sparse inputs enable CPU-friendly nets

Future Directions

- Deeper search with better pruning
- Time management optimization
- Train stronger NNUE models

Thank you!