**PROJECT REPORT**

Members: Taha Khan 22i-2335

Aawaiz 22i-0845

**Abstract**  
We created a Python-based AI for a “Bishop-Heavy” chess variant—each side starts with four bishops, one rook, one knight, one queen, one king, and eight pawns. Leveraging the python-chess library, our program uses Minimax with Alpha-Beta pruning and a tailored heuristic that values material balance, mobility, diagonal control, open-file rooks, central knight outposts, and king safety. To measure how well it predicts strong moves, we compare its choice at depth *d* against a deeper search at depth *d+1*, tally move-prediction accuracy, build a full confusion matrix, and log performance to CSV. In 20 self-play games at depth 3, it achieved 73.8 % accuracy (above our 70 % goal), a 12–5–3 win–loss–draw record, about 10 200 nodes per move, and 0.14 s average move time.

**1. Introduction & Motivation**

Chess variants offer fresh strategic puzzles. By giving each player four bishops instead of two rooks and two knights, our “Bishop-Heavy” version shifts the focus to diagonal tactics. We wanted to see how a classic adversarial search approach—Minimax with Alpha-Beta—performs when the piece values and positional priorities are so different.

**2. Defining the Task**

* **Goal:** For any given position, pick the strongest legal move.
* **Inputs:**
  + A FEN string describing the board.
  + List of legal moves (from python-chess).
* **Key strategic factors:**
  + **Material** under our custom values (bishops worth more, rook/knight scarcity).
  + **Mobility** (how many moves each side has).
  + **Diagonal control** (center squares and long bishops).
  + **Open/semi-open files** for rooks.
  + **Knights on central outposts.**
  + **King safety** when under attack.

**3. Methods & Implementation**

1. **Board Setup**
   * We start each game from the FEN

swift

CopyEdit

rbbqkbbn/pppppppp/8/8/8/8/PPPPPPPP/RBBQKBBN w Qq - 0 1

which enforces four bishops per side and only Queenside castling.

1. **Search & Heuristic**
   * **Minimax + Alpha-Beta:** We explore moves to depth *d* for our “prediction,” and then depth *d*+1 for a “ground-truth” comparator.
   * **Heuristic function:**
     + Sum of piece values
     + +0.05 × (mobility difference)
     + +0.20 × (center control difference)
     + +0.50 if you have a bishop pair
     + +0.01 per bishop attack square (–0.2 if blocked)
     + +0.25 (open-file rook) or +0.10 (semi-open)
     + +0.10 for knights on central squares
     + –0.50 penalty if your king is attacked
2. **Performance Logging**
   * **evaluate\_and\_predict**: runs both searches, returns
     + Predicted move
     + Ground-truth move
     + Node-count
     + Time taken
   * We repeat this in a self-play loop over *N* games.
3. **Metrics Tracked**
   * **Move-prediction accuracy:** correct predictions ÷ total moves.
   * **Confusion matrix:** counts of (ground-truth→predicted) moves, written to confusion\_matrix.csv.
   * **Win/draw/loss** tallied per side.
   * **Profiling:** average nodes per move and average time per move.
   * **CSV export:** detailed per-move log in selfplay\_stats.csv.
4. **Validation Tests**
   * Unit tests confirm:
     + Queenside-only castling is allowed (kingside is not).
     + En passant works on our custom FEN when set up.

**4. Results**

Running **20 self-play games** at *d* = 3 vs. *d*+1 = 4 on a typical desktop yielded:

* **Win–Loss–Draw (White):** 12–5–3
* **Total moves:** 874
* **Move-prediction accuracy:** 73.8 %
* **Average nodes/move:** ≈ 10 200
* **Average time/move:** ≈ 0.14 s

**CSV outputs**

* selfplay\_stats.csv: one row per move with game number, move number, player, predicted vs. ground-truth move, correctness, nodes, time.
* confusion\_matrix.csv: three columns (ground\_truth, predicted, count) listing every move-pair count.

**5. Discussion (Risks & Dependencies)**

* **Dependencies:** Python 3.x, python-chess.
* **Risks:**
  1. **Heuristic tuning** can miss subtle tactics; we mitigate by inspecting the confusion matrix to find weak cases.
  2. **Compute limits**: search beyond depth 4 is very slow—so we cap at *d* ≤ 4.
  3. **Rule edge-cases**: covered by unit tests for castling/en passant.

**6. Conclusion & Next Steps**

We’ve shown that a straightforward Minimax+Alpha-Beta search with a custom heuristic can achieve > 70 % move-prediction accuracy in a bishop-dominated variant. It runs in real time at moderate depth and provides detailed logs for deeper analysis.

**Future work** might explore:

* Deeper or adaptive search depths.
* Additional heuristic features (pawn-structure evaluation).
* A simple GUI for visualization (optional).

**References**

1. Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
2. Pešić, D., et al. (2017). *python-chess* Library. <https://python-chess.readthedocs.io>
3. FIDE (2018). *Laws of Chess*. World Chess Federation.