# PROJECT REPORT

HUMAN VS CHESS AI

**Group members:**

* Haris Akhtar 221357
* Kanwar Junaid 221414
* Jasim Ibrahim 221390

**Submitted to:** Miss Warda Aslam



**Department Of Computer Science**

**Air University Islamabad (AU)**

### 1. Project Overview

This project implements a rule-enforcing, intelligent Chess AI system using Python. It features a fully interactive GUI built with Tkinter and a multi-difficulty AI that combines core Artificial Intelligence techniques: **Genetic Algorithm**, **Minimax with Alpha-Beta Pruning**, and **constraint-based rule enforcement**. The AI offers three levels of play: random (easy), evolved (medium), and optimized (hard). The system ensures legal gameplay, including complex chess rules like castling, en passant, and checkmate detection — all built from scratch without external chess libraries.

### 2. Motivation

Most chess engines prioritize raw computational depth (e.g., Stockfish), often at the cost of explainability and adaptability. Our goal was to design a lightweight AI that demonstrates multiple AI paradigms while enforcing all chess rules independently. We aimed for a balance between:

* Speed and intelligence
* Rule completeness
* AI diversity (randomness, evolution, optimization)

This project serves as a pedagogical example of how different AI strategies perform in a constrained, complex domain like chess.

### 3. Objectives

* Design a **modular chess engine** enforcing all core chess rules manually.
* Implement three difficulty levels:
  + Random (easy)
  + Genetic Algorithm-based evolution (medium)
  + Minimax + Alpha-Beta Pruning (hard)
* Develop a **Tkinter-based GUI** for real-time, human-vs-AI interaction.
* Incorporate a **comprehensive board evaluation function**.
* Log AI performance (nodes searched, move decision time).

### 4. Scope

* Human vs AI gameplay (single-player)
* Three AI levels: Random, Genetic Algorithm, Alpha-Beta Pruning
* Full rule enforcement: check, checkmate, castling, en passant, pawn promotion
* Custom evaluation: material, mobility, king safety, pawn structure
* Interactive Tkinter GUI

### 5. AI Techniques Used

#### 5.1 Random Move Selection (Easy Mode)

* Chooses a random legal move from available options.
* Intended as a baseline for comparison with smarter modes.

#### 5.2 Genetic Algorithm (Medium Mode)

* **Population-based** move generation using random mutation.
* **Tournament selection** to choose high-fitness candidates.
* **Fitness function** scores based on board evaluation.
* 5 generations per move; diversity ensures creative responses.

#### 5.3 Minimax with Alpha-Beta Pruning (Hard Mode)

* Searches decision tree recursively up to fixed depth (3).
* Uses **alpha-beta pruning** to skip non-optimal branches.
* Highly deterministic and optimal within its depth limit.
* Performance metrics include node count and response time.

### 6. Evaluation Function

A handcrafted evaluation function scores a given board state using:

|  |  |
| --- | --- |
| Factor | Description |
| Material | Weighted value of remaining pieces |
| Positional Bonuses | Center control, piece mobility |
| King Safety | Bonus for castled king, penalty for exposed king |
| Pawn Structure | Penalizes doubled, isolated, and backward pawns |
| Mobility | Number of legal moves available |
| Special States | Bonus for castling rights, penalty for vulnerable king |

* **GUI (Tkinter):** Handles clicks, rendering, and move animations.
* **Game Logic:** Validates moves, enforces turns, manages game state.
* **AI Engine:** Generates moves based on selected difficulty.

### 7. Implementation Highlights

* **Piece-specific move generation** (king, rook, bishop, etc.)
* Special move support: castling, en passant, promotion
* Turn-based GUI with visual feedback
* AI performance monitoring (nodes evaluated, execution time)
* Clean code modularity (board, player, AI modules)

### 8. Evaluation & Testing

|  |  |
| --- | --- |
| Metric | Observation |
| Move Legality | 100% rule-compliant |
| Easy Mode (Random) | Instant response, poor play quality |
| Medium Mode (GA) | 1–2s delay, creative but not optimal |
| Hard Mode (Minimax) | 2–4s delay, strong play |
| GUI Performance | Smooth and responsive |
| Rule Coverage | Full (castling, en passant, promotion) |

### 10. Conclusion

This project demonstrates how multiple AI techniques can be meaningfully integrated in a strategic game like chess. By combining **Genetic Algorithms**, **Minimax with Alpha-Beta Pruning**, and a **comprehensive rule system**, we created a robust AI capable of responding to various skill levels. The GUI adds real-time interactivity and educational value. Overall, the system fulfills both its academic and functional objectives.

### 11. References

* Stuart Russell & Peter Norvig, Artificial Intelligence: A Modern Approach
* Tkinter GUI Docs – <https://docs.python.org/3/library/tkinter.html>
* Genetic Algorithm Concepts – Analytics Vidhya, TutorialsPoint
* Alpha-Beta Pruning – GeeksForGeeks, MIT AI Notes
* Python OOP Best Practices