**Applied Artificial Intelligence**

**Assignment - 3: Chess Game**

**Course Instructor**

Dr. Atif Jilani

**Submitted by**

Abdullah Daoud……………………………………………………….(22I-2626)

**Section**

SE-E

**Date**

**Wednesday, April 16th, 2025**

**Spring 2025**



**Department of Software Engineering**

FAST – National University of Computer & Emerging Sciences

Islamabad Campus

Table of Contents

[**1. Introduction 3**](#_heading=h.681pn5nuvqob)

[**2. Board Initialization 3**](#_heading=h.shtz7jcs35dq)

[**3. Move Generation 3**](#_heading=h.stkslxm1m8r4)

[**4. Move Execution 3**](#_heading=h.7y9zenes4gyj)

[**5. Check and Game State Detection 3**](#_heading=h.agw2t7i7owuc)

[**6. AI Decision-Making 4**](#_heading=h.9iimomuy8nu7)

[**7. Game Loop and User Interaction 4**](#_heading=h.e8cf70qolq92)

[**8. Results Obtained 4**](#_heading=h.pyzu6lopd1k5)

[**9. Known Issues (Based on Implementation Review) 5**](#_heading=h.ryvsfo39q1hs)

[**10. Conclusion 5**](#_heading=h.ofn46qyrlw6f)

# 1. Introduction

This report provides a brief explanation of the steps involved in the provided Python chess game program and summarizes the results obtained from its functionality. The program implements a console-based chess game with a human player (white) and an AI opponent (black), supporting all standard chess rules, including castling, en passant, promotion, checkmate, and stalemate.

# 2. Board Initialization

* **Description**: The ChessBoard class initializes an 8x8 chessboard with the standard starting position. White pieces (uppercase: P, N, B, R, Q, K) are placed on ranks 1 and 2, black pieces (lowercase: p, n, b, r, q, k) on ranks 7 and 8, and empty squares are marked with '.'. Additional attributes track the current player, move history, en passant target, castling availability, and move counters.
* **Key Methods**: \_\_init\_\_, print\_board
* **Purpose**: Sets up the game state and provides a visual representation of the board.

# 3. Move Generation

* **Description**: The program generates legal moves for each piece type (pawn, knight, bishop, rook, queen, king) based on chess rules. Each piece type has a dedicated method (e.g., get\_pawn\_moves, get\_knight\_moves) that computes possible moves, considering captures, special moves like castling, and en passant. The get\_legal\_moves\_for\_piece method filters moves to ensure they don’t leave the player’s king in check.
* **Key Methods**: get\_legal\_moves, get\_legal\_moves\_for\_piece, get\_pawn\_moves, get\_knight\_moves, get\_bishop\_moves, get\_rook\_moves, get\_king\_moves
* **Purpose**: Ensures only valid moves are available, adhering to chess rules.

# 4. Move Execution

* **Description**: The move\_piece method updates the board by moving a piece from a start to an end position. It handles special cases like castling (moving the rook), en passant (capturing a pawn), and promotion (replacing a pawn with a queen, rook, bishop, or knight). It also updates castling availability, en passant targets, and move counters.
* **Key Methods**: move\_piece, get\_piece
* **Purpose**: Applies player and AI moves to the board, maintaining game state integrity.

# 5. Check and Game State Detection

* **Description**: The program checks if the current player’s king is in check (is\_in\_check), if the position is checkmate (is\_checkmate), or stalemate (is\_stalemate). The is\_square\_attacked method determines if a square is under attack by opponent pieces, used to validate moves and detect checks.
* **Key Methods**: is\_in\_check, is\_square\_attacked, is\_checkmate, is\_stalemate
* **Purpose**: Detects critical game states to determine when the game ends (win or draw).

# 6. AI Decision-Making

* **Description**: The ChessAI class implements an AI opponent using the Minimax algorithm with alpha-beta pruning. The evaluate\_position method assigns a score based on material (piece values), center control, pawn structure, mobility, and king safety. The AI selects the best move by exploring the game tree up to a depth of 3.
* **Key Methods**: evaluate\_position, minimax, get\_best\_move
* **Purpose**: Enables the AI to play competitively as black, making strategic moves.

# 7. Game Loop and User Interaction

* **Description**: The main function runs the game loop, alternating between human (white) and AI (black) turns. It displays the board, prompts for human moves in algebraic notation (e.g., e2e4), validates input, and handles promotions. The AI computes its move automatically. The game ends on checkmate, stalemate, or if the user quits.
* **Key Methods**: main, pos\_to\_notation, notation\_to\_pos
* **Purpose**: Manages the flow of the game and user interaction.

# 8. Results Obtained

The chess game program functions as intended, with the following outcomes based on its implementation:

* **Correct Board Setup**: The game starts with a correctly initialized chessboard, displayed clearly in the console with algebraic notation (a-h, 1-8).
* **Move Validation**: The program accurately generates and validates moves for all piece types, supporting:
  + Standard moves (e.g., e2e4 for pawns, g1f3 for knights).
  + Special moves like castling (e1g1 moves king to g1, rook to f1), en passant, and pawn promotion (prompts for Q/R/B/N).
  + Move filtering to prevent illegal moves (e.g., moving into check).
* **Game State Detection**:
  + Correctly identifies check (e.g., when a king is attacked).
  + Detects checkmate (e.g., Scholar’s Mate: e4, e5, Bc4, Qh5, Qf7#).
  + Recognizes stalemate (e.g., when a player has no legal moves and is not in check).
* **AI Performance**: The AI plays reasonable moves, avoiding repetitive patterns due to an improved evaluation function that considers material, position, and mobility. It responds to common openings (e.g., e4 e5, Nf3 Nc6) and can deliver checkmate in simple positions.
* **User Interaction**: The interface is user-friendly, accepting moves in standard notation, providing clear error messages for invalid inputs (e.g., “Illegal move. Try again.”), and allowing the user to quit.
* **Robustness**: The program handles edge cases like invalid inputs, promotions, and draw conditions without crashing.

# 9. Known Issues (Based on Implementation Review)

* **Move Validation Efficiency**: The get\_legal\_moves\_for\_piece method uses deep copies for each move to check for check, which could be optimized for performance in complex positions.
* **AI Strength**: The AI’s depth of 3 limits its strategic depth, potentially missing deeper tactics, though sufficient for casual play.
* **Halfmove Clock Accuracy**: The halfmove clock reset condition may need adjustment to ensure compliance with FIDE rules for captures (currently checks post-move state).

# 10. Conclusion

The chess game program successfully implements a playable chess experience with full rule compliance. It provides a solid foundation for casual play, with accurate move generation, game state detection, and a competent AI opponent. Future improvements could focus on performance optimization and enhancing AI strength for more challenging gameplay.