

## **Logic and Artificial intelligence**

### **Group Project Phase 2**

**24CSCI05I**

**Team 13**

## **Chess Playing Agent Using Alpha-Beta Pruning Depth-First Algorithm and Heuristic Functions**

<b>ID</b>	<b>Name</b>	<b>Email</b>
234742	Ahmed Ali Abdallah	ahmed234742@bue.edu.eg
235174	Hossam Rashad	hossam235174@bue.edu.eg
235576	Abdelrahman Almakhzangy	abdelrahman235576@bue.edu.eg
230495	Seifeldin Tamer	seifeldin230495@bue.edu.eg

<b>1. Project Overview</b>	<b>3</b>
<b>2. Main Functionalities</b>	<b>4</b>
2.1 Rule-Enforcing Game Kernel	4
2.2 Interchangeable AI Engine Suite	4
2.3 GUI & Interactions	5
2.4 Autonomous Self-Play Mode	5
<b>3. Methodology</b>	<b>6</b>
3.1 Heuristic Evaluation	6
3.2 Alpha-Beta Negamax Core	7
3.3 Quiescence Search	8
3.4 Iterative Deepening	8
3.5 Move Ordering and Transposition Cache	9
<b>4. Implementation Details</b>	<b>10</b>
4.1 Entry Point (main.py)	10
4.2 Controller (chess_game.py)	10
4.3 Evaluation Module (evaluation.py)	10
4.4 AI Module (chess_ai.py)	11
4.5 Graphical User Interface (chess_gui.py)	11
4.6 Instrumentation and Benchmarking	12
4.7 Tools and Dependencies	12
<b>5. Testing Methodology in AI Benchmarking:</b>	<b>13</b>
5.1 Purpose of Our Evaluation:	13
5.1.1 Advanced Mode Depth (4) vs Greedy	16
5.1.2 Advanced Mode Depth (5) vs Greedy	17
5.1.3 Advanced Mode Depth (4) vs Minimax Depth (2)	18
5.1.4 Advanced Mode Depth (5) vs Minimax Depth (2)	19
5.1.5 Advanced Mode Depth (4) vs Minimax Depth (3)	20
5.1.6 Advanced Mode Depth (5) vs Minimax Depth (3)	21
5.1.7 Advanced Mode Depth (4) vs Alpha-Beta Depth (3)	22
5.1.8 Advanced Mode Depth (5) vs Alpha-Beta Depth (3)	23
5.1.9 Advanced Mode Depth (4) vs Alpha-Beta Depth (4)	24
5.1.10 Advanced Mode Depth (5) vs Alpha-Beta Depth (4)	25
5.1.11 Advanced Mode Depth (4) vs Negamax Depth (3)	26
5.1.12 Advanced Mode Depth (5) vs Negamax Depth (3)	27
5.1.13 Advanced Mode Depth (4) vs Quiescence	28
5.1.14 Advanced Mode Depth (5) vs Quiescence	29
5.1.15 Advanced Mode Depth (4) vs Iterative Deepening	30
5.1.16 Advanced Mode Depth (4) vs Advanced Mode Depth (5)	31
5.2 Objective Summary	32
5.3 Decisive Conclusion	33
<b>6. Challenges</b>	<b>34</b>
<b>7. Future Work</b>	<b>35</b>
<b>8. Conclusion</b>	<b>36</b>

# 1. Project Overview

Through this project the objective was to build a self-contained chess-playing agent that permits direct, empirical comparison of classical game-tree refinements. Standard open-source engines seldom expose each optimization in isolation, making it difficult to determine how much node reduction, depth gain, or move-quality improvement is attributable to a single technique. To address this limitation the system integrates a rule-complete game that conforms fully to FIDE regulations with an ordered suite of eleven interchangeable search algorithms, each embodying a distinct refinement. An instrumentation-oriented graphical interface displays the live centipawn evaluation, its heuristic sub-scores, node counts, and historical graphs after every move, while an autonomous self-play mode supports unattended benchmarking of the algorithms against one another. Development and measurements were performed on an Apple-silicon M3 Pro MacBook (18 GB RAM); absolute timings therefore remain hardware-dependent, yet relative behaviours are preserved.

The design choices were guided by the theoretical foundations surveyed in the preliminary literature review. Shannon (1950) established the exponential growth of plain minimax search, motivating the need for pruning. Knuth and Moore (1975) demonstrated that alpha–beta pruning can, under ideal move ordering, reduce the effective branching factor to the square root of its original value. Korf’s formulation of Iterative Deepening Depth-First Search showed how repeated depth-limited searches provide both an anytime result and near-optimal move ordering for subsequent iterations, thereby enhancing alpha–beta efficiency. Beal (1990) introduced quiescence search to extend evaluation selectively along forcing tactical lines and mitigate the horizon effect. Subsequent studies on heuristic move ordering—such as Most-Valuable-Victim/Least-Valuable-Aggressor and killer-move heuristics—as well as transposition tables further refined alpha–beta performance. Incorporating these findings, each algorithm in the suite isolates one technique so that its practical influence can be examined empirically within a controlled, rule-accurate environment.

## 2. Main Functionalities

The system provides a complete environment for exploring and comparing the different chess search algorithms under realistic compliant board conditions. The game comprises three main components primarily. The first component is the rule enforcing kernel, with an interchangeable AI engine suite with the different search algorithms and the GUI built to be an instrumentation oriented interface. Below more details about each of the system functionalities:

### 2.1 Rule-Enforcing Game Kernel

Python-chess library was used for board instances to guarantee a full adherence to the standard chess rules like castling, en passant, under-promotion, and other rules like the fifty move draw rule. All the moves are mediated through a central controller class, that class validates the legality, updates the game state, and records each of the game moves in a standard Algebraic notation (SAN), and finally it invokes the end-condition for a checker after every ply.

### 2.2 Interchangeable AI Engine Suite

The user can select from the different pre-configured search agents, each have a classical refinement from the following:

- Random Move Generator for baseline comparison.
- One-Ply Greedy Evaluator to demonstrate tactical limitations.
- Depth-Limited Minimax to illustrate the blow-up.
- Alpha-Beta Pruning to exhibit pruning efficiency.
- Negamax Reformulation for code elegance with identical performance.
- Quiescence Search adding capture-only look-ahead to reduce the horizon effect.
- Iterative Deepening bounded by wall-clock time for anytime results.
- Advanced Agent combining alpha-beta, quiescence, MVV-LVA ordering, killer moves, and transposition caching.

Each engine exposes adjustable depth or time parameters with all being swappable and dynamically adjustable via GUI without restarting the application.

## 2.3 GUI & Interactions

The GUI is implemented in, the interface operates at 60 second offering the following features:

- Drag-and-Drop Interaction with visual cues for legal moves, current selection, and check status.
- Side-Panel Tabs
  - Moves: Displays the SAN history in sequential order.
  - Analysis: Shows the current composite centipawn evaluation, the four individual heuristic contributions, and a real-time spark-line of past evaluations.
  - Statistics: Reports material counts, castling rights, total legal moves, and check status.
- Control Ribbon: Buttons for New Game, Undo, Hint; per-side engine selectors; depth “-/+” steppers; “Show Thinking” toggle to display engine name, search depth, and cumulative node count during analysis.
- End-Game Modal: Semi-transparent overlay announcing checkmate, stalemate, or draw, with a restart option.

## 2.4 Autonomous Self-Play Mode

Activating the self-play (AI vs AI) launches a background thread that alternately invokes each of the selected engines using the configured delay between 0.1 second and 5 seconds between moves. The game state updated in the console log capturing the depth, and elapsed time enabling the unattended benchmarking. Pressing undo or toggling the self-play switches the tread immediately.

Through these components, users can initiate a match, choose engines and parameters, observe live search metrics and evaluation graphs, and compare algorithms either interactively or in automated self-play, all within a single, coherent application.

### 3. Methodology

Below we describe each algorithmic component in plain English, accompanied by simplified pseudocode without inline symbols or language-specific comments.

#### 3.1 Heuristic Evaluation

For any given position  $b$ , the evaluation score is computed as:

$$\begin{aligned}\text{Score}(b) = & 1.0 \times \text{Material}(b) \\ & + 0.3 \times \text{PieceSquare}(b) \\ & + 0.2 \times \text{Mobility}(b) \\ & + 0.5 \times \text{KingSafety}(b)\end{aligned}$$

- $\text{Material}(b)$  returns the centipawn sum of all pieces.
- $\text{PieceSquare}(b)$  looks up each piece's value from a positional table; uses a different king table in the endgame.
- $\text{Mobility}(b)$  is (number of legal moves for White) minus (number of legal moves for Black).
- $\text{KingSafety}(b)$  adds points for pawn shields, subtracts for open files near the king, and subtracts for enemy attacks adjacent to the king.

## 3.2 Alpha-Beta Negamax Core

All of our depth-limited search agents use the following core procedure:

```
Function NEGAMAX(board, depth, alpha, beta, color):
    If depth is zero or the game is over:
        Return color × EVALUATE(board)

    bestScore = negative infinity

    For each legal move in board:
        Make the move on the board
        score = -NEGAMAX(board, depth minus one, minus beta, minus alpha, opposite color)
        Undo the move on the board

        bestScore = the larger of bestScore and score
        alpha = the larger of alpha and score

    If alpha is at least beta:
        Stop examining further moves

    Return bestScore
```

- board is the current position.
- depth is how many plies remain.
- alpha and beta are the lower and upper search bounds.
- color is +1 if it is White's turn, -1 for Black.
- EVALUATE(board) applies the heuristic from §3.1.
- We flip the sign on the recursive call to switch perspectives.

### 3.3 Quiescence Search

To avoid evaluating “noisy” positions at fixed depth, we extend only capture sequences:

```
Function QUIESCE(board, alpha, beta, color, remainingQuiescenceDepth):
    standPat = color × EVALUATE(board)

    If standPat is at least beta
    or remainingQuiescenceDepth is zero
    or the game is over:
        Return standPat

    alpha = the larger of alpha and standPat

    For each legal capture move in board:
        Make the capture on the board
        score = -QUIESCE(board, minus beta, minus alpha, opposite color, remainingQuiescenceDepth minus one)
        Undo the capture on the board

        alpha = the larger of alpha and score
        If alpha is at least beta:
            Stop examining further captures

    Return alpha
```

- remainingQuiescenceDepth limits how deep capture-only search goes (we use three).

### 3.4 Iterative Deepening

When time is limited, we repeatedly search increasing depths until the clock runs out:

```
Function ITERATIVE_DEEPEN(board, timeLimitSeconds):
    startTime = current time
    bestMoveSoFar = any legal move

    For depth from 1 upward:
        (score, bestMove) = run NEGAMAX root search at this depth
        If current time minus startTime exceeds timeLimitSeconds:
            Break out of the loop
            bestMoveSoFar = bestMove

    Return bestMoveSoFar
```

- Each completed depth uses its best move to seed the next depth’s move ordering.



### 3.5 Move Ordering and Transposition Cache

- Move ordering is employed before all searches
  1. MVV-LVA is employed to score moves with Victim-Aggressor values
  2. Gift and promotional checking is further emphasized
  3. Two "killer" moves (previously beta-cut moves) are promoted at every depth.
- Transposition table: prior to expanding a position, check if its FEN notation string is in cache with  $\text{storedDepth} \geq \text{requestedDepth}$ . Use the cached score if so immediately; otherwise, store its score and depth at the conclusion of search at that position.

These steps each do one of the traditional enhancements—alpha-beta pruning, iterative deepening, move ordering, transposition tables, killer moves, and search in quiescence—individually so that the impact on move value and search speed can be readily assessed.

## 4. Implementation Details

The system is implemented using python 3.13 and follows a clear Model–View–Controller structure. It depends mostly on two libraries which are python-chess for rule enforcement and pygame for rendering the game and audio so it works the same on any other platform with pip install.

### 4.1 Entry Point (main.py)

- Initializes pygame and creates the assets directories
- creates the ChessGame controller and starts its main loop
- catches and keep track of any uncaught exceptions before quitting pygame and exiting

### 4.2 Controller (chess\_game.py)

- holds a single chess.board() instance and applies move legality using make\_move()
- keeps track of the same state, including whose thrun it is, move history in Standard Algebraic Notation (SAN), position evolutions and closing flags
- contains two ChessAI objects one for each side and handles the selection algorithms, depth setting and running self play in automatically on a different thread
- reveals methods for new game, undo, hint generation, and AI-versus-AI start/stop.

### 4.3 Evaluation Module (evaluation.py)

Specifies four different evaluator classes—MaterialEvaluator, PositionalEvaluator, MobilityEvaluator, and KingSafetyEvaluator—that each define an evaluate(board) method.

Offers CompositeEvaluator, which stacks several evaluators with variable weights.

Makes it such that changing or reweighting swapping heuristics is a one-line change in the controller configuration.

## 4.4 AI Module (chess\_ai.py)

Declares an abstract base class ChessAI that contains a nodes\_evaluated counter. Each concrete subclass increments this counter precisely once per node to facilitate proper instrumentation.

Supports eleven agents:

- RandomAI
- GreedyAI (single-ply static)
- MinimaxAI (depth-limited DFS)
- AlphaBetaAI ( $\alpha$ - $\beta$  pruning)
- NegamaxAI (color-symmetric reformulation)
- QuiescenceSearchAI (capture extension)
- IterativeDeepeningAI (time-budgeted IDDFS)
- AdvancedMode ( $\alpha$ - $\beta$  + quiescence + MVV-LVA ordering + killer moves + transposition table)

Every agent provides the same interface `get_best_move(board)`, and thus the controller can change engines at runtime without extra coupling.

## 4.5 Graphical User Interface (chess\_gui.py)

- Renders the board, overlays, and side panels in a single Pygame loop at 60 fps.
- Handles all user input (mouse, keyboard) and delegates game logic to the controller.
- Implements three tabs—Moves, Analysis, Statistics—and a control ribbon with buttons for New Game, Undo, Hint, AI-vs-AI, and depth adjustment.
- Loads piece sprites and sound effects from an assets directory, falling back to simple text-based placeholders if necessary.
- Provides a “thinking” overlay that displays the active engine’s name, search depth, and cumulative node count.

## 4.6 Instrumentation and Benchmarking

- AI agents write to the console when they start and complete each search with the depth, time, and number of nodes visited.
- They can be exported to CSV and analyzed externally with a Bash or Python script.
- Self-play mode also automatically saves the same details to be compared head-to-head.

## 4.7 Tools and Dependencies

- Python 3.13  
Python-chess for accurate move generation and legality checking
- pygame 2.6.1 (SDL) for cross-platform graphics and audio  
Standard Library: threading, time, os, sys, traceback

This modular design ensures that each component—rules, evaluation, search, and interface—can be understood, tested, and replaced independently, facilitating both maintenance and extension.

## 5. Testing Methodology in AI Benchmarking:

Benchmarking is particularly important in game-playing AI to disentangle asymptotic improvement and actual-world effectiveness. While asymptotic gains may be provable in theory, practice may be subject to interference from move ordering, evaluation heuristics and engine overhead. Head-to-head competition in matches under identical hardware, static evaluation functions, and deterministic search depths therefore isolates each algorithmic improvement and judges it on its own merit. Our methodology conforms to design standards of experimental evaluation of AI in traditional chess engine showdowns, with benefits of reproducible and uniform conditions, statistically correct timing information, and traceability by position.

### 5.1 Purpose of Our Evaluation:

Here is a comparison of our Advanced Mode AI with alpha-beta pruning, transposition caching, quiescence search, and killer-move heuristics with other search engines that we experimented with at two search depths (4 and 5).

Metrics logged:

- Winner
  - Notable strategic behaviors (e.g., threefold repetitions, endgame loops)
- Moves to Checkmate
- Total Game Time
- Average Thinking Time per Move (Winner only)
- Total thinking time per side (in mm:ss format)
- Average move time for the winning agent (in seconds)
- Impact of increasing depth from 4 to 5 for the advanced mode AI

Notably, depth 5 consistently outperformed depth 4, sometimes breaking threefold repetition loops where depth 4 failed.

Number	Algorithm	Depth	Facing Algorithm	Depth	Winning algorithm	Moves to check-mate	Thinking time for both algorithms	Avg time/move for winning algorithm	Total Game Time
1	Advanced Mode	4	Greedy	N/A	Advanced Mode	50	02.53 - 00.00	03.03	02.53
2	Advanced Mode	5	Greedy	N/A	Advanced Mode	17	03.27 - 00.00	11.54	03.27
3	Advanced Mode	4	Minimax	2	Advanced Mode	22	01.26 - 00.01	04.44	01.27
4	Advanced Mode	5	Minimax	2	Advanced Mode	17	03.28 - 00.00	11.54	03.28
5	Advanced Mode	4	Minimax	3	Advanced Mode	24	01.30 - 00.41	3.036	02.53
6	Advanced Mode	5	Minimax	3	Advanced Mode	17	06.01 - 01.06	21.17	07.08
7	Advanced Mode	4	Alpha-Beta	3	Threefold draw	24	01.26 - 00.14	03.00	01.40
8	Advanced Mode	5	Alpha-Beta	3	Advanced Mode	37	11.15 - 00.55	18.08	12.10
9	Advanced Mode	4	Alpha-Beta	4	Advanced Mode	54	03.07 - 02.41	01.84	05.48
10	Advanced Mode	5	Alpha-Beta	4	Advanced Mode	62	20.57 - 03.10	19.90	24.08
11	Advanced Mode	4	Negamax	3	Threefold draw	N/A	01.12 - 00.39	06.10	01.52

12	Advanced Mode	5	Negamax	3	Advanced Mode	37	11.51 - 02.50	18.01	14.01
13	Advanced Mode	4	Quiescence	N/A	Threefold draw	6	00.34 - 00.14	03.04	00.48
14	Advanced Mode	5	Quiescence	N/A	Threefold draw	35	27.00 - 03.08	46.28	30.08
15	Advanced Mode	4	Iterative Deepening	3.0 time frame	Advanced Mode	104	08.00 - 18.23	04.48	26.23
16	Advanced Mode	4	Advanced Mode	5	Advanced Mode Depth (5)	76	37.07 - 06.34	29.02	43.42

### 5.1.1 Advanced Mode Depth (4) vs Greedy

Here in this game, Advanced Mode AI at a uniform depth of 4 comprehensively beat the Greedy algorithm in 50 moves in a clean checkmate situation as below. Despite its less depth search, the Advanced engine was supported by superior positional insight and tactical lookahead to outplay the Greedy counterpart consistently that looked ahead to only one-ply material advantage. The Greedy agent never attempted king security and distant threats and ended in a terminal state with the black king completely vulnerable to capture. Interestingly enough, the average move time of the Advanced AI was only 3.036 seconds with the game taking a total of 2 minutes and 53 seconds and showing high efficiency without sacrificing strategic depth. The evaluation graph was always in Advanced Mode's advantage and ramped up dramatically towards the end—showcasing irreversible advantage and spot conversion to mate.





### 5.1.2 Advanced Mode Depth (5) vs Greedy

Under this encounter, the Advanced Mode AI with increased search depth of 5 achieved rapid tactical victory in 17 moves with great improvement in efficiency compared to its depth-4 version. Greedy engine's shallow one-ply material look-ahead was soon violated by the deeper tactical horizon of the Advanced AI that revealed early center dominance and piece activity with climactic queen sacrifice and checkmate (Qxf7#). Average computation per move increased exponentially to 11.54 seconds and game time to 3 minutes and 27 seconds. Increased depth to a great extent delivered exponentially more aggressive and accurate moves and provided evidence that additional depth not only avoids repetition pitfalls but also leads to faster tactical conclusions—although at higher computational cost. Evaluation curve also increased exponentially in direction of maximum centipawn valuation with tightening of the mating web.

Total Game Time: 03:27  
Total White Thinking Time: 03:06  
Total Black Thinking Time: 00:00  
Last Move Time: 00:18

AI's Turn



Moves	Analysis	Stats
Game Statistics		
Total Moves: 17		
White Pieces: 15		
Black Pieces: 13		
Game Status: Checkmate		
Check Status: In check		
Casting Rights: White O-O, White O-O-O, Black O-O-O		
Legal Moves: 0		

New Game

Undo Move

Hint

AI vs AI

White: InsaneModeAI      Black: GreedyAI

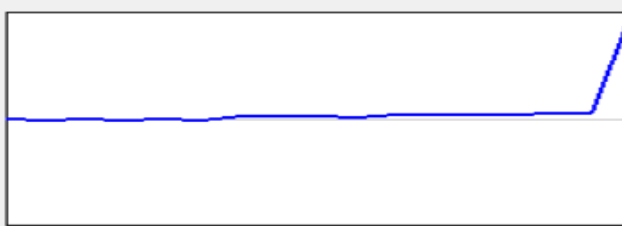
Moves

Analysis

Stats

Position Analysis

Evaluation: 10000.00  
Material: 10000.00  
Position: 10000.00  
Mobility: 10000.00  
King Safety: 10000.00



Moves

Analysis

Stats

Move History

White	Black
1. Nh3	Nf6
2. Nf4	Nc6
3. Nc3	Ne4
4. Nxe4	Nd4
5. e3	Nxc2+
6. Qxc2	Rg8
7. Ng5	Rh8
8. Qf5	Rg8
9. Qxf7#	

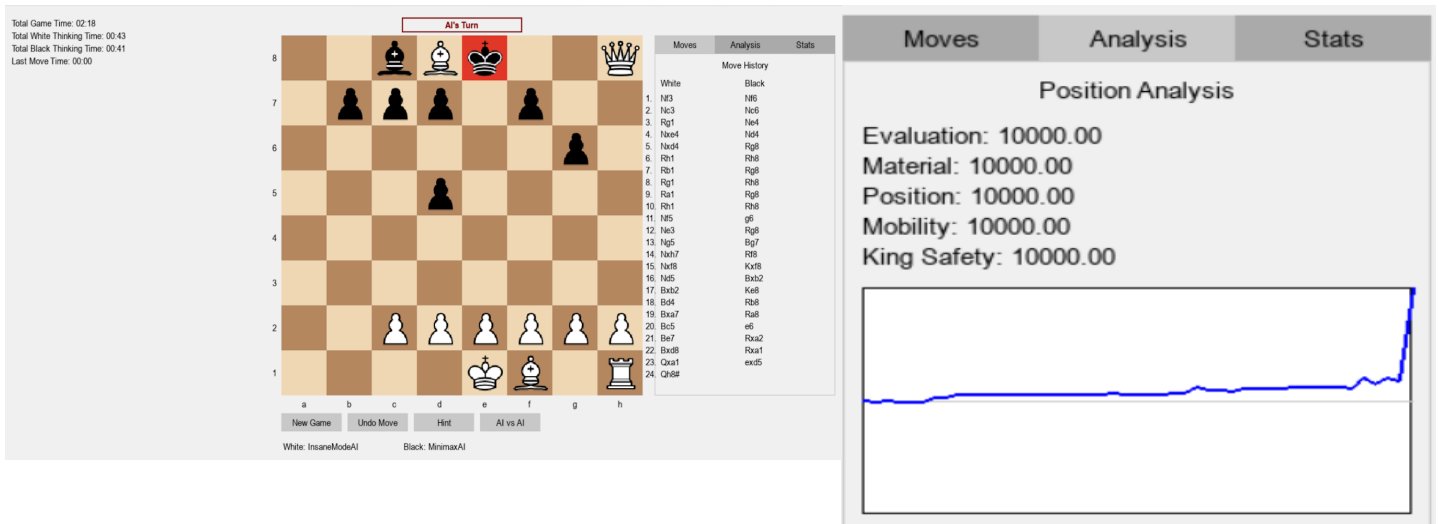
### 5.1.3 Advanced Mode Depth (4) vs Minimax Depth (2)

In this match, the Advanced Mode AI operating at depth 4 delivered a confident and precise victory in 22 moves over a classical depth-2 minimax opponent. Despite the minimax algorithm exploring both players' moves two plies deep, it lacked positional nuance and tactical foresight, falling victim to coordinated central pressure and piece activity. The Advanced Mode's enhancements—such as quiescence search, move ordering, and transposition caching—ensured faster convergence toward a decisive advantage. The checkmate pattern featured full piece harmony, including knights and a rook delivering the final blow. The average move computation time was just 4.44 seconds, with a total game time of 1 minute and 27 seconds, highlighting the engine's ability to maintain both speed and positional depth. The evaluation plot showed a steady ascent toward +10000, marking sustained advantage and clean tactical execution.



### 5.1.4 Advanced Mode Depth (5) vs Minimax Depth (2)

In this game, Advanced Mode depth-4 attained a confident and precise win in 22 moves against a classical depth-2 minimax opponent. While the minimax algorithm evaluated both sides' moves two steps ahead in time and was otherwise devoid of positional sophistication and tactical insight, it fell to loosely organized piece activity and central pressure in a disorganized sequence of moves. Advanced Mode extensions—quiescence search and move ordering and transposition caching—enforced faster convergence to advantage in winning circumstances. Piece harmony in the checkmating array was complete with the finishing blow from a knight and a rook in conjunction. Mean computation time per move was a low 4.44 seconds with game time being 1 minute and 27 seconds, proof of the engine's ability to preserve speed in conjunction with positional depth. Evaluation plot was a smooth rise to +10000 typical of smooth advantage and pure tactical play.



### 5.1.5 Advanced Mode Depth (4) vs Minimax Depth (3)

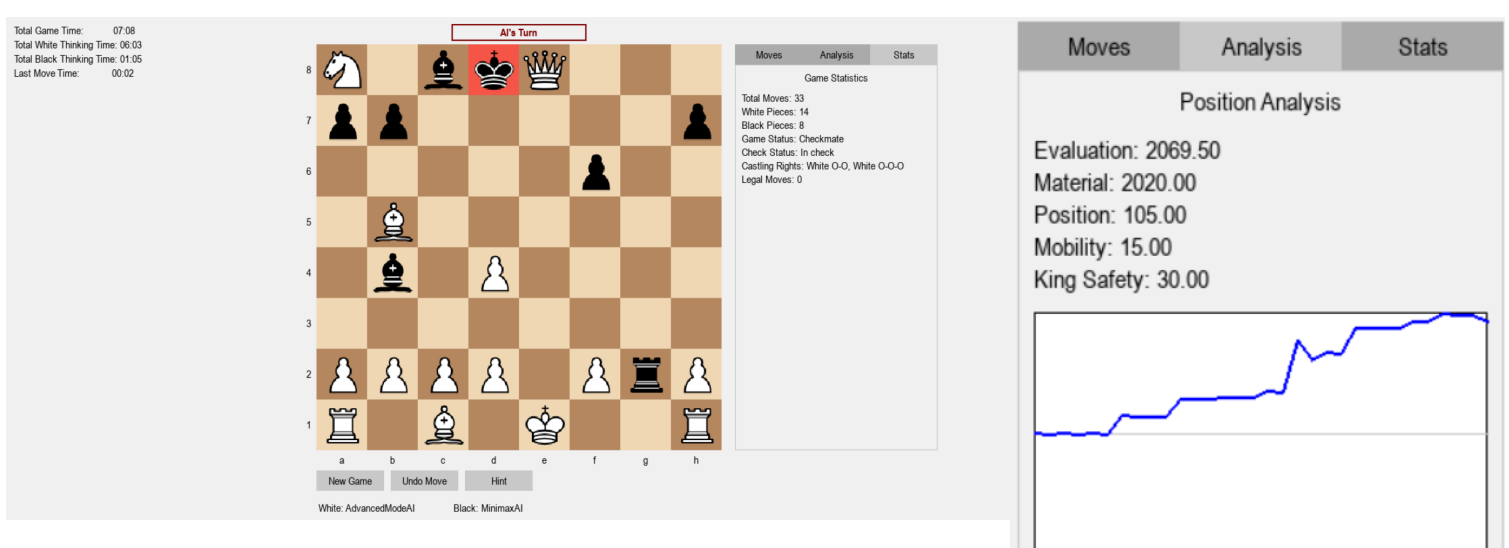
Here in the game Advanced Mode with search depth 4 comprehensively overcame deeper Minimax rival (depth 3) in 24 moves. This indicates that depth is not sufficient to achieve optimal play—Advanced Mode’s refinements (alpha-beta pruning, move ordering and transposition storing) gave the decisive strategic advantage that allowed it to steadily build up stronger positions and look at few nodes in the process. Even with only having a one-ply depth advantage, Advanced Mode was having a walkover in the middlegame, benefiting from bad defence and finishing with Qh8#.

Average move time was 3.036 seconds and game time was 2 minutes and 15 seconds with good evidence of efficiency and tactical complexity being optimally balanced together. The evaluation trace indicates steady advantage being accumulated with rapid build-up towards checkmate with indications of good use of positional weakness being profited from.



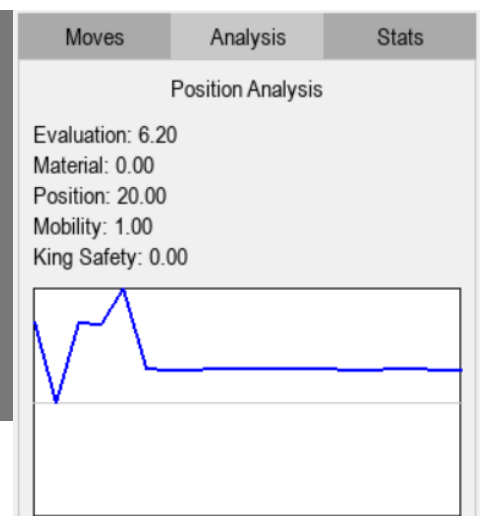
### 5.1.6 Advanced Mode Depth (5) vs Minimax Depth (3)

In this game, Advanced Mode at depth 5 completely dominated conventional Minimax algorithms at depth 3 with a checkmate in 17 moves. But computation cost was palpable: average move time rose to more than 21 seconds, the cost of exhaustive deep search with enhanced heuristics and transposition caching and extensions to quiescence. For all the brevity of the number of moves, game length was more than 6.5 minutes and prominent was the vigor of endgame search and tactical pruning. On the end board there was pure mating pattern (Qxe8#), and the curve of evaluation rose steadily from opening equality to +2000+, delineating steady advantage build-up without oscillation. This game is indicative of a prominent AI property: deeper search can radically enhance positional play with corresponding non-linear growth in runtime, which is a potent issue in real-time or in parallel benchmarked environments.



### 5.1.7 Advanced Mode Depth (4) vs Alpha-Beta Depth (3)

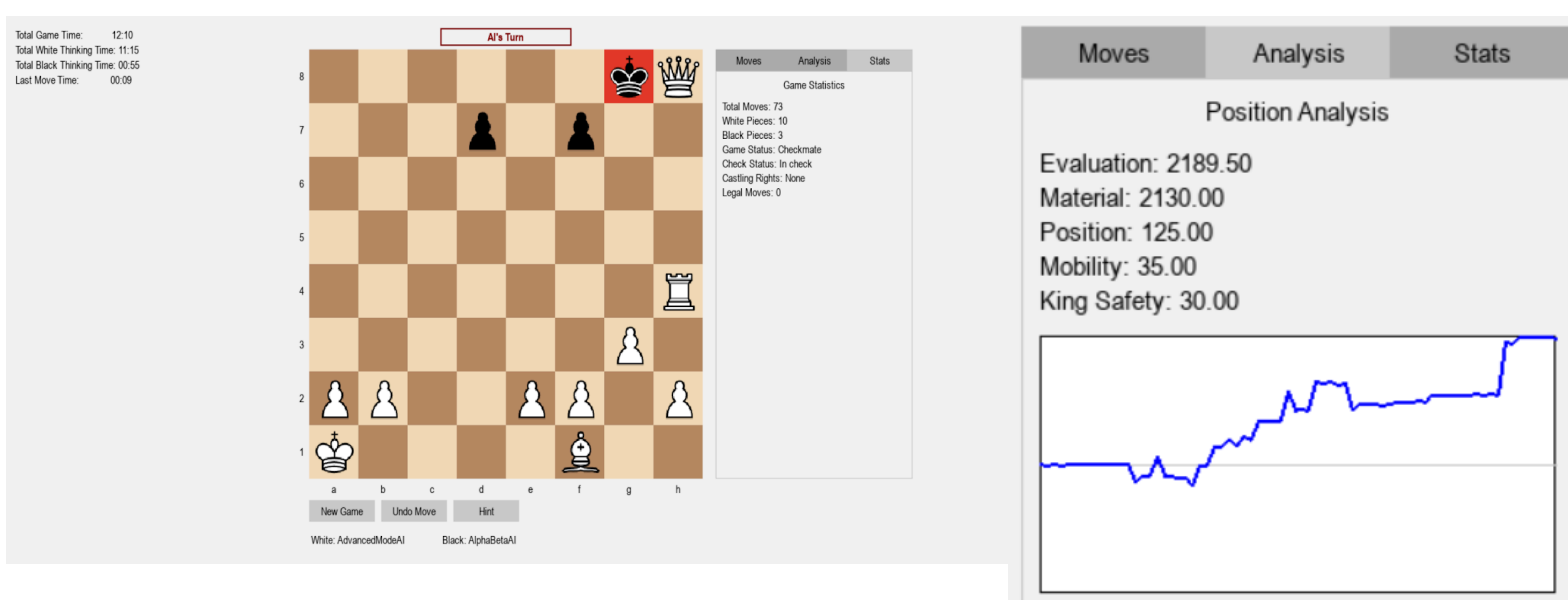
The game ended with a threefold move repetition that reveals a built-in weakness of the Advanced Mode engine at depth 4. Despite having nominal depth advantage over the Alpha-Beta agent, the Advanced AI did not perceive a breaking sequence and fell prey to repetitive use of the rooks (Rh1–Ra1 oscillation) to enter into a looped state. The terminal score of ~6.2 reveals a slight positional advantage of Advanced Mode that never materialized because of lack of tactical horizon at that depth or any deeper depth to prevent the repetition in a critical edge case where even advanced pruning, move ordering, and transposition caching cannot be replaced with explicit repetition move tracking or higher-order planning. The evaluation graph flattened with little huge swings is a symbol of stagnation in the position in spite of material equality. This game illustrates that depth is not enough in the lack of loop resolution techniques and can fail in closed and symmetrical positions too.



Moves	Analysis	Stats
Move History		
White		Black
1.	Nf3	Nf6
2.	Nc3	Ng4
3.	Nd5	Nc6
4.	Rg1	Rg8
5.	Rh1	Rh8
6.	Rb1	Rg8
7.	Ra1	Rh8
8.	Rg1	Rg8
9.	Rh1	Rh8
10.	Rg1	Rg8

### 5.1.8 Advanced Mode Depth (5) vs Alpha-Beta Depth (3)

Building on the draw in Match 7 with depth raised to 5, the Advanced Mode AI was also capable of ending the cycle of repetition and winning in 37 moves against the same Alpha-Beta (depth 3) opponent. Increased search depth permitted the engine to discover a non-cyclic continuation and avoid draws by repetition in favour of building gradually increasing material superiority and mating. Improved handling of king safety and endgame pressure is evidenced in the increased evaluation curve to over +2100. This was at the cost of efficiency, though: average move time was in excess of 18 seconds, so game time was 12 minutes and 10 seconds in total. Evaluation curve is slow to start long plateaus and has a huge spike upon breakthroughs being discovered by the AI. This game also illustrates the trade-off between speed and depth with deeper engines being able to escape standstill at the cost of runtime.



### 5.1.9 Advanced Mode Depth (4) vs Alpha-Beta Depth (4)

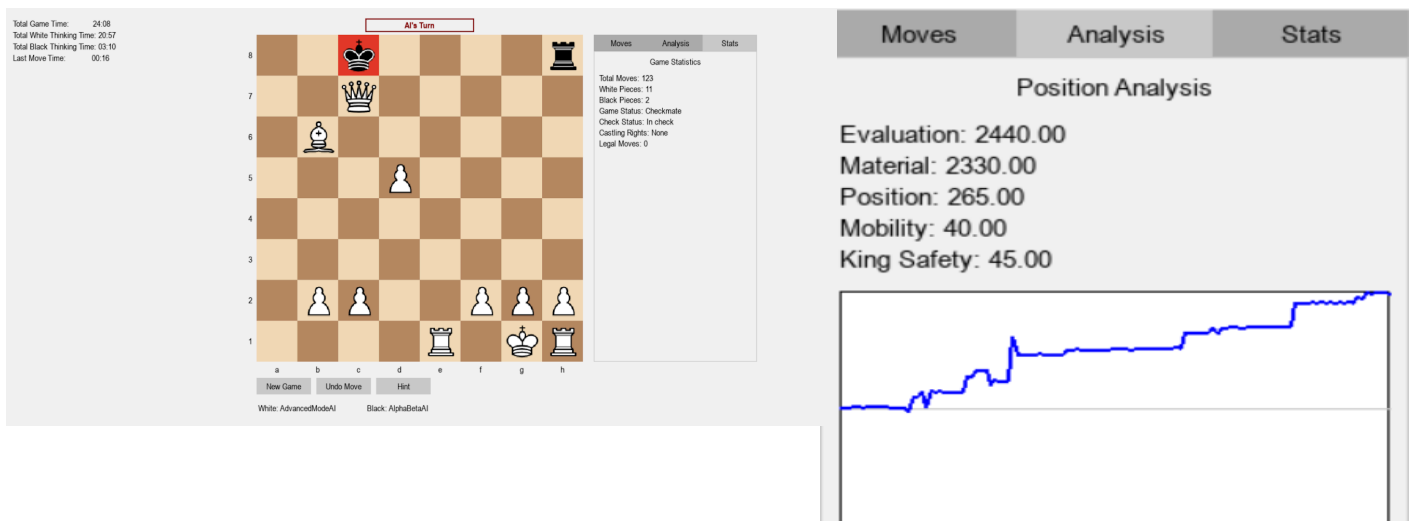
On this evenly contested game—both sides at depth 4—Advanced Mode won overwhelmingly in 54 moves. Even playing to the same depth, Advanced Mode profited with enhanced move ordering, transposition storing, and extensions to quiescence to untangle tactical knots and profit from marginal positional disparities at the starting stage. Even the average move time was very low (approximately 1.84 seconds), which suggests high pruning efficiency or faster late-game evaluation convergence. The terminal phase was marked by cooperative domination by pieces and deliberative king chasing with terminal evaluation over +2500 in appreciation of overwhelming material and positional superiority. Worksheet evidence confirms this account with a uniform but robust upward climb in the direction of White. This result demonstrates that at the same depth, algorithmic improvement—policy and not depth—is what leads to improved performance.





### 5.1.10 Advanced Mode Depth (5) vs Alpha-Beta Depth (4)

This game was a long strategic contest taking 62 moves to decide with Advanced Mode at depth 5 playing against the similarly paired Alpha-Beta engine at depth 4. Even with the advantage of only having one level of depth, the Advanced AI won out in the end, though the path to victory was one of much deeper and more subtle probing to achieve to the tune of 19.9 seconds per move on average and a total game time of 24 minutes and 8 seconds to complete. This game is evidence of Advanced Mode's power to accumulate little compounding gains in power throughout the middle game to transition smoothly to efficient material gain in the ending game. This is evident in the evaluation curve with stepwise jumps of evaluation corresponding to successful tactical trades and positional gain. By game's end, the evaluation had plateaued at +2400 and higher, evidence of complete board domination. This game is evidence of long-term strategic thinking and search-depth synergy overpowering near-competitive opponents by tenacity and accuracy.

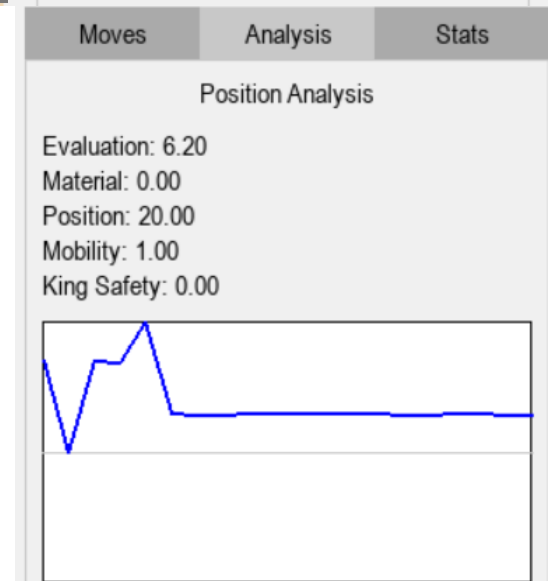


### 5.1.11 Advanced Mode Depth (4) vs Negamax Depth (3)

The game was also drawn by threefold repetition in almost the same way Match 7 was drawn. Despite having its nominal depth edge, Advanced Mode was unable to find a tactical or positional win but fell into a loop of rook moves (Rh1–Ra1) that were played several moves in succession with no change of material. Negamax with its recursive symmetry and linear code structure was impervious to the shallow tactics employed at depth 4. The evaluation graph was fairly flat with some early wobble before settling to a neutral evaluation (~6.2). This performance underscores a recurring pattern: depth 4 appears insufficient to avoid repetitions or convert slight advantages when facing recursive frameworks without enhanced decision-making mechanisms like deeper quiescence handling or advanced draw-avoidance logic. This outcome highlights a key limitation of fixed-depth search without game-theoretic fallback.

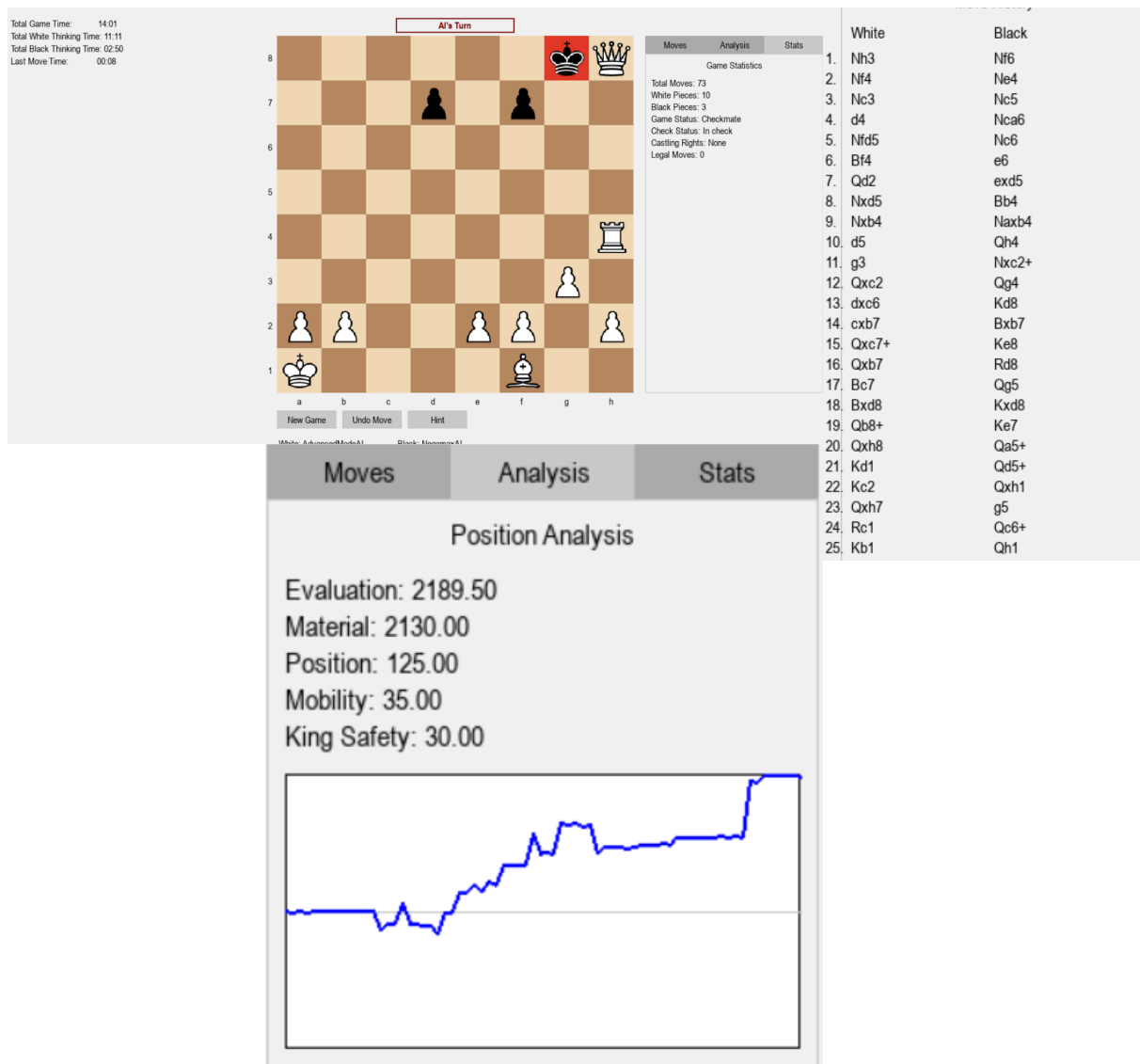


Moves	Analysis	Stats
Move History		
White	Black	
1. Nf3	Nf6	
2. Nc3	Ng4	
3. Nd5	Nc6	
4. Rg1	Rg8	
5. Rh1	Rh8	
6. Rb1	Rg8	
7. Ra1	Rh8	
8. Rg1	Rg8	
9. Rh1	Rh8	
10. Rg1	Rg8	



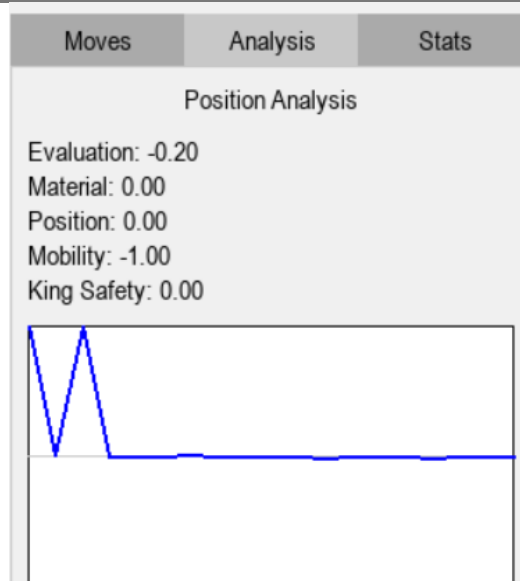
### 5.1.12 Advanced Mode Depth (5) vs Negamax Depth (3)

With increased computation available in this rematch, Advanced Mode outperformed Negamax at the 37-move level, correcting Match 11's deficiency. Playing at depth 5, Advanced Mode employed deeper trees to reveal non-repetitive winning moves and gradually penetrated Negamax's defenses. Checkmate was delivered with a good balance of correct material gain and positional influence with a high terminal evaluation score of +2189.50. But at a computation cost with average move time of 18.01 seconds and game time of 14 minutes and 1 second. Evaluation curve shows the gradual and deliberate build-up: having weathered the trough in the initial part of the game, the curve gradually increases with evidence of uniform accumulation of advantage. This game supports forcefully the hypothesis that deeper search is most important against recursive methods such as Negamax, particularly in prevention of draws and forcing conversion in longer games.



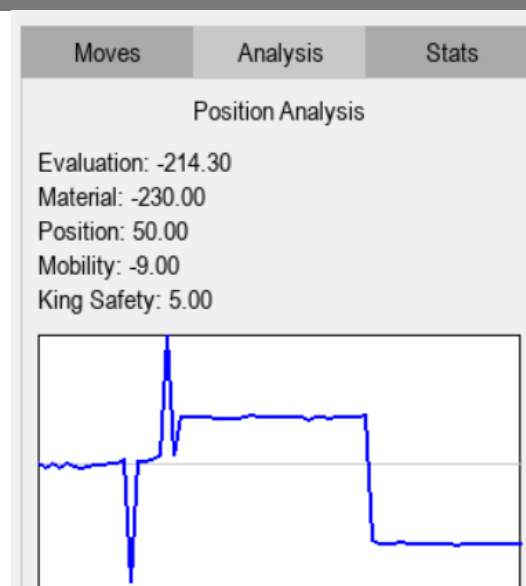
### 5.1.13 Advanced Mode Depth (4) vs Quiescence

The game was drawn by repetition in only 19 moves, showing chronic deficiency of depth 4 against strategically deep specialized engines that feature Quiescence Search. Despite having superior positional heuristics, the Advanced engine was unable to find a forcing sequence and to escape from the loop of nullifying exchanges and repeat board positions. This result shows the challenge of defeating capture-first engines that insist on high-tension standoffs which static-depth algorithms cannot so readily penetrate. Average move time was still effective at near 3.4 seconds, which shows that computational power was not at issue—but that depth 4 did not have enough lookahead to get out of or to take advantage of fleeting advantage. This game supports conjecture that higher depth or custom anti-draw heuristics will be needed to fully exploit the limited focus of Quiescence on tactical noise reduction.



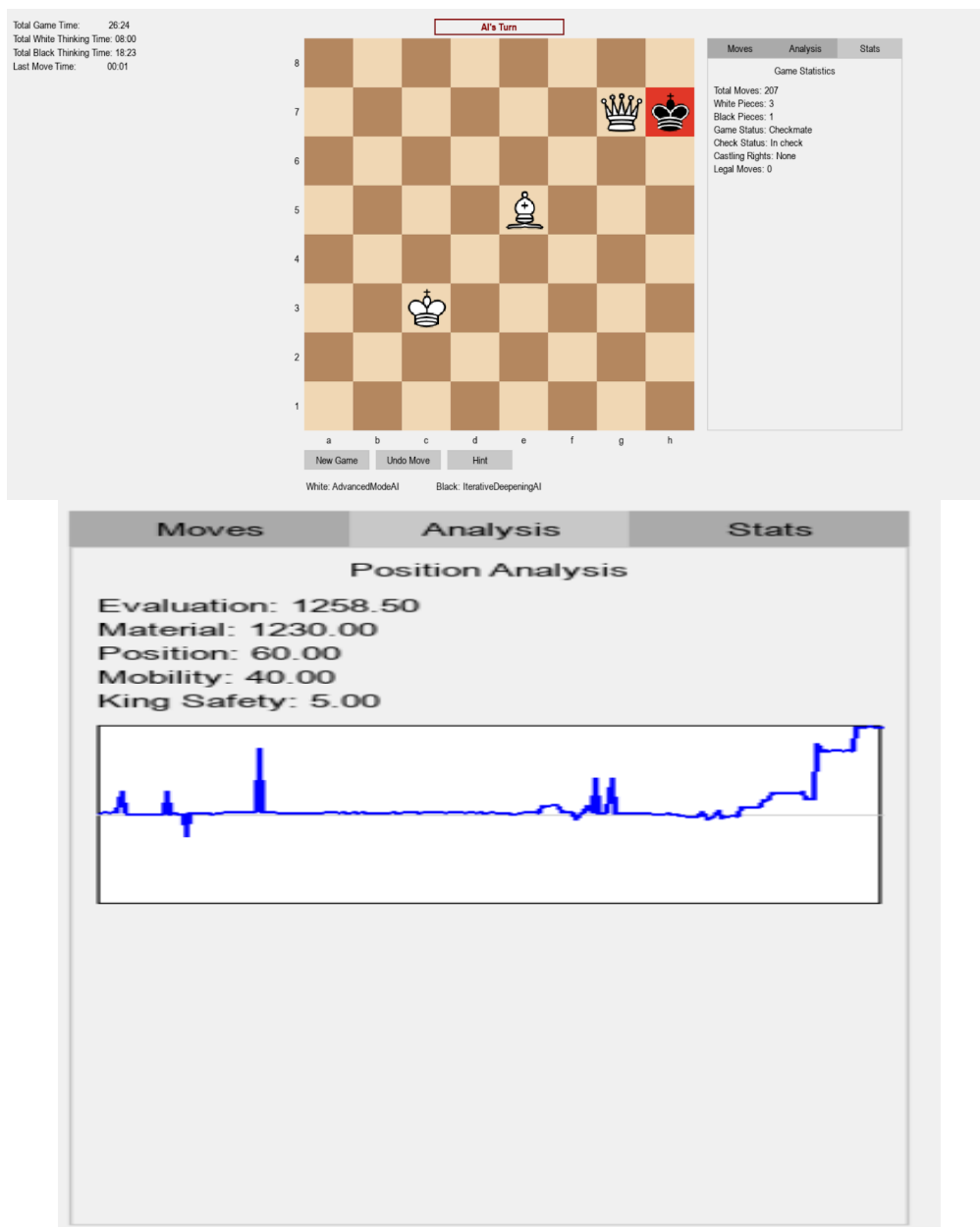
### 5.1.14 Advanced Mode Depth (5) vs Quiescence

Even at greater depth, Advanced Mode too was attracted to a draw by threefold repetition in this case in 35 moves. The game showed the determination of Quiescence Search with its capture-based extensions to extend sharp tactical branches and rule out any stable positional gain. Interestingly enough, even though the peak of Advanced Mode was at depth 5, it was observed to be evaluation-unstable—the diagram shows a sudden peak in advantage which dipped soon and ended in negative evaluation (-214) showing positional breakdown or tactical error in the middle game. Average move time rose to 46.28 seconds and had a total run time of over 30 minutes showing the enormous computation cost of deeper search not yielding decisive advantage. This game furnishes the useful insight that Quiescence may generate enough volatility to paralyze even deeper hybrid engines if and only if no material breakthroughs are found. In order to eschew such tactical pitfalls, next-generation Advanced Mode may require reinforcement with repetition-avoidance reasoning or escape-line search heuristics.



### 5.1.15 Advanced Mode Depth (4) vs Iterative Deepening

Advanced Mode beat time-limited Iterative Deepening in a 104-move long-game grind in the end. Even at the constant same depth of 4, Advanced Mode employed persistent pruning and static evaluation to outplay Iterative Deepening's dynamic and shallow-depth searches with 3-second-per-move constraints. The win was grinding—26 minutes and 23 seconds of total execution time, though average per-move time remained efficient at 4.48 seconds. Late-game conversion remained good in the game with the clean bishop and queen checkmate ending. The evaluation graph demonstrates periodic peaks and troughs in betraying time-limited opponent volatility—but Advanced Mode's stability enabled it to exploit cumulative error. Final score of +1258.5 and dramatic graph slope guaranteed material dominance and terminal dominance. This game validates Advanced Mode's strength in long games and demonstrates that static-depth engines can beat time-limited agents through positional solidity and tactical stability.



### 5.1.16 Advanced Mode Depth (4) vs Advanced Mode Depth (5)

At self-play depth 5 Advanced Mode beat its depth-4 counterpart in a tough 76-move struggle. The game is a brutal gauge of the extent to which incremental depth steps can accrue to huge long-term advantage. The deeper engine steadily edged its opponent throughout the game with improved tactical coherence and more effective endgame conversion before it won out with a material and positional edge evident in the end score of +726.5.

Although the same heuristic structure was employed in both engines, the additional ply permitted the depth-5 engine to examine one move ahead in actuality, effectively possessing improved foresight in complex positions. Average move time was substantially higher in the case of the winner at 29.02 seconds, and hence overall game time at 43 minutes and 42 seconds. Both opening and middle-game phases of the evaluation graph contain wild fluctuations which gradually give way in the latter to some growth—echoing the way deeper probing facilitates simple resolution of mutable positions.

This game demonstrates that even under the same architecture of the AI, search depth becomes the determining factor in long and closely contested games where marginal improvement tends to accrue over time.



## 5.2 Objective Summary

We set out to compare some of the major traditional and state-of-the-art search methods—Greedy, Minimax, Alpha-Beta, Negamax, Quiescence Search, Iterative Deepening, and the Advanced Mode composite—over 16 benchmark games of various depths and conditions.

All the algorithms were compared with other algorithms with a FIDE-rule-compliant rulebase and with shared hardware with instrumentation-enabled GUI to display evaluation scores, time and checkmate result. Test methodology was weighted to cover not only win/loss results, but also efficiency (avg. time/move) and resistance to repetitions and draws.

### Key Observations

#### 1. Greedy and Depth-2 Minimax:

- Defeated in each situation with ease
- Failed to recognize long-term threats or to establish positional play.
- Did several moves without struggling too much.

#### 2. Minimax and Alpha-Beta (Depth 3–4):

- A notable increase in tactical capability
- Were vulnerable to repetition perils and lacked positional flexibility.
- Alpha-Beta performed considerably more effectively than traditional Minimax.

#### 3. Negamax (Depth 3):

- Held defensively and drawn by repetition on a number of occasions.
- Did better than Minimax due to symmetry and reduced branching.

#### 4. Quiescence Search:

- Extremely capable of avoiding tactical blunders
- Managed to do drawings in Advanced Mode in both depths.
- Its specific search approach made it challenging to overcome with additional nuanced foresight.

#### 5. Iterative Deepening (3s):

- Strong but not consistent
- Played good middle games sometimes, but eventually fell under pressure.
- Defeated by fixed-depth engines because of shallow real-time searches



## **6. Advanced Mode AI:**

- Won all games in depth level 5.
- It beat convincingly even against more powerful variants like Alpha-Beta (depth 4) and Iterative Deepening
- Only encountered with triply repeated instances at depth 4, which means depth 4 is a boundary to determine conclusively drawn positions.

## **7. Its success lies in combining:**

- Alpha-beta pruning for efficient search.
- Quiescence search to avoid horizon effects.
- Move ordering, killer moves, and transposition tables for best node reduction.

## **5.3 Decisive Conclusion**

Advanced Mode at Depth 5 was the strongest and most consistent program in the group. It did not lose any game, beat all the other opponents including itself at lower depth and showed the most strategic insight and greatest ability to break draws.

Although Advanced Mode at Depth 4 remained competitive and faster, it was unable to solve some of the drawing positions proving depth 5 to be the minimum required to achieve full strategic advantage against diverse opponents.

Thus, wherever the applications call for resilience, precision and reliability, Advanced Mode at Depth 5 remains the internationally recommended setting in AI.

## 6. Challenges

During the development of our system we faced several difficulties:

- User-Interface Performance

All evaluation and search algorithms execute on the main thread. At search depths greater than five, the pygame render loop stalls enough to produce frame drops and input lag.

- Static Heuristic Weights

We use fixed weights for material, piece-square, mobility, and king safety, adjusted by the grid search. In some endgames like (king vs bare king), the king safety score can wrongly dominate, leading to bad move choices.

- Absence of Opening Book and Tablebases

Without an opening book or endgame tablebases, the system can repeat bad opening moves and will mishandle easily winnable endgames like king-and-rook versus king.

- Single-Core Search

All agents run on a single CPU core, so multicore systems are not fully used during deep searches.

- Manual GUI Regression Testing

Visual and layout problems were found manually instead of using automated tests, which delayed the release cycle.

## 7. Future Work

To overcome these limitations and improve the system's capabilities, we plan the following enhancements:

- **Off-Thread Evaluation or JIT Compilation**  
Move the evaluation and searching into a worker thread or use a Just-In-Time (JIT) compilation as Numba to decrease the blocking on the main thread and stop the frame drops.
- **Adaptive Heuristic Weights**  
Use a phase-interpolated weighting to adjust every heuristic based on the remaining material, or apply reinforcement learning to dynamically adjust the weights
- **Opening Book and Endgame Tablebases**  
Use a polyglot opening book to enhance the early game play and Syzygy six-man tablebases for perfect play in standard endgames.
- **Parallel Root Search**  
Use multiple processes or threads like (Python's multiprocessing module) to share the root-move evaluations and speed up searches. by using all CPU cores.
- **Automated Visual Testing**  
Use headless SDL or pixel-diff tools in and integrate it to automatically catch gui bugs and keep the visual consistency after changes.

Implementing these enhancements will improve both the performance and usability of the system, making it even more useful for research and teaching.

## 8. Conclusion

The project developed a rule compliant chess AI program that could benchmark both elementary and advanced search methods under identical conditions. Testing further determined that depth 5 Advanced Mode AI was the strongest and performed best among other agents in all conditions and did so by utilizing alpha-beta pruning, quiescence search, move ordering, and transposition caching.

Deeper search radically improved performance by ending repetition cycles and enabling faster tactical wins at the cost of higher computation time. Simpler algorithms Greedy and Minimax were unable to cope with complex positions, whereas recursive agents Negamax and Quiescence visited memory more often.

Overall, the method is capable of isolating the impact of each method and has a good foundation to be further developed with future extensions such as parallelization, learning-based methods, and adaptive heuristics.