AI Assignment-3

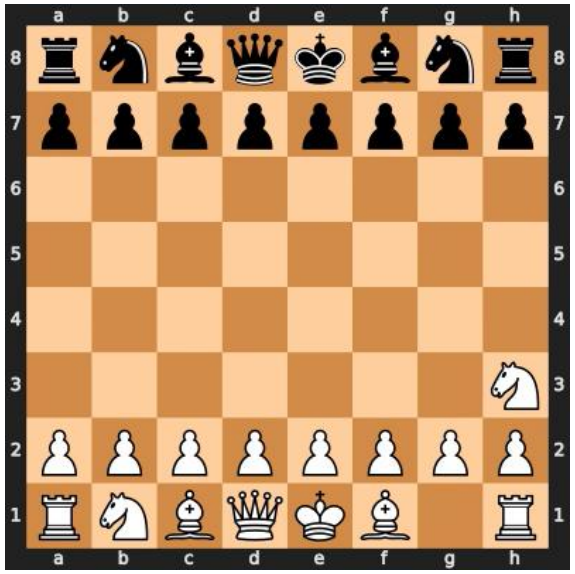
Implementations

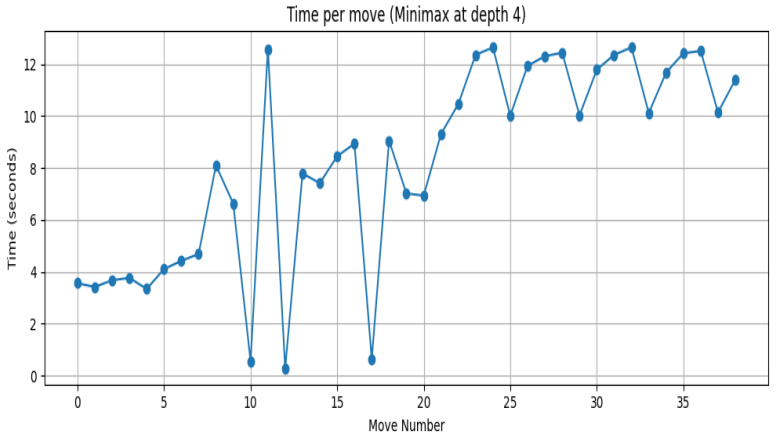
- Minimax and αβ-pruning algorithms on the Chess environment.

Team Members

- Dharmendra Chauhan (CS24M115)
- Kodela Phanindra(CS24M121)

Minimax Algorithm on the Chess Environment





Average time (for first 10 moves) = 3.9 sec
Max time overall = 12.7 sec
Average time (for last 10 moves) = 11.8 sec

Evaluation function used:

Material-based evaluation:
Pawn = 1, Knight = 3, Bishop = 3, Rook = 5, Queen = 0 and King = 0.
Score = White's total – Black's total

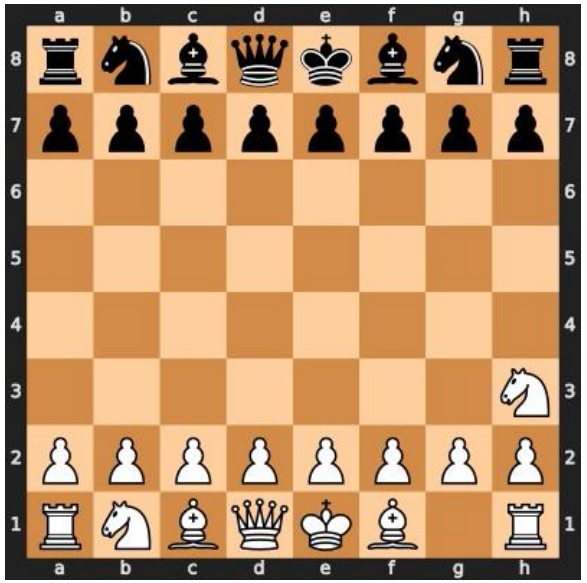
Observation:

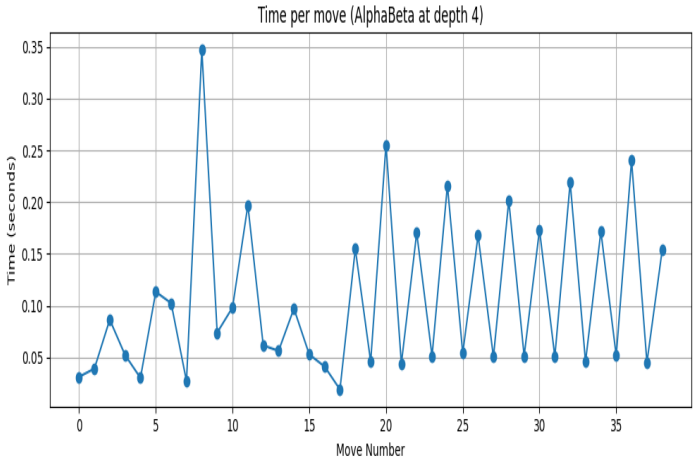
Time per move increases significantly as the game progress, especially after mid-game, likely due to the growing number of possible legal moves and deeper search trees, making later decisions more computationally expensive.

Characteristics:

1. *Decision-making for two-player games* – it helps an AI choose the best move by thinking ahead about all possible moves and counter-moves.
2. *Maximizes win, minimizes loss* – One player tries to get the highest score (max), the other tries to stop them (min).
3. *Assumes both play perfectly* – It picks moves based on the idea that both players will always make the best possible move.

aß-pruning Algorithm on the Chess Environment





Average time (for first 10 moves) = 0.12 sec
Max time overall = 0.35 sec
Average time (for last 10 moves) = 0.16 sec

Evaluation function used:

Material-based evaluation:
Pawn = 1, Knight = 3, Bishop = 3, Rook = 5, Queen = 0 and King = 0.
Score = White's total – Black's total

Observation:

Time per move remains consistently low and stable throughout the game, with all moves taking under 0.35 seconds, demonstrating the significant efficiency improvement over plain Minimax by pruning unnecessary branches.

Characteristics:

1. *It speeds up the Minimax algorithm* by skipping moves that won't affect the final decision (pruning unnecessary branches).
2. *Uses two values (Alpha and Beta)* to keep track of the best options for each player and decide when to stop checking further.
3. *Finds the same best move as Minimax*, but does it much faster by checking fewer possibilities.

Performance analysis b/w Minimax and AlphaBeta on the Chess Environment

Situation	Preferred Algorithm	Why
Time Efficiency	AlphaBeta	Significantly lower average and maximum time per move
Scalability (Late Game)	AlphaBeta	Maintains consistent performance regardless of move number
Complex Game States	AlphaBeta	Handles increasing complexity without major time spikes
Raw Decision Quality	Minimax	Evaluates all nodes at depth, potentially considering more options

Efficiency Tradeoff

1. AlphaBeta offers *over 95% reduction in move computation time,* with max_time = 0.35s vs 12.7s in Minimax.
2. Minimax explores *all possible states at a given depth*, making it slower but thorough, useful when computational resources are not a constraint

Key Observation

1. *AlphaBeta remains efficient* even in the late-game phase where Minimax time spikes drastically.
2. *Minimax shows instability* with erratic spikes and dips in mid-game, indicating inconsistent search complexity.