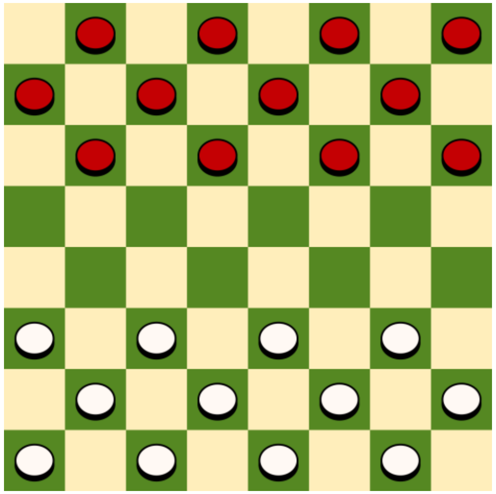
AI Checkers Game

Informed & Uninformed Search • BFS • Minimax

RULES AND REGULATIONS

•**Board**: 8×8 grid, dark squares used for play.
•**Players**: Two sides — *Black* and *White*
•**Starting Setup**: Each player places 12 pieces on the dark squares of the first 3 rows on their side.
•**Objective**: Capture all opponent's pieces or block them so they can't move.
•**Moves**:
•Pieces move **diagonally forward** one square to an empty square.
•Capturing is done by **jumping over** an adjacent opponent piece to an empty square immediately beyond it.
•Multiple jumps in a single turn are allowed if possible.
•**Captures are mandatory** (must take if available).
•**Kings**:
•A piece reaching the opponent's back row is **kinged** (crowned).
•Kings can move **diagonally forward and backward**.
•**Winning**: Opponent has no legal moves (all pieces captured or blocked).



Overview

- Simplified Checkers implementation.
- Two modes:
- - Uninformed Search (BFS)
- - Informed Search (Minimax + Alpha-Beta)
- Player (Black) vs Computer (White).
- Depth limit: 4.

Uninformed Search (BFS)

- Breadth-First Search explores moves level-by-level.
- No evaluation of board quality — just legality.
- Selects first valid move at depth.
- Heuristic_uninformed: piece count (+1 normal, +2 king).
- Pros: Simple. Cons: Weak gameplay.

Informed Search (Minimax)

- Simulates moves ahead up to depth 4.
- Alternates maximizing (computer) and minimizing (player).
- Uses Alpha-Beta pruning to cut irrelevant branches.
- Heuristic_informed:
- - Piece weights + king bonus
- - Center control
- - Mobility advantage.

Heuristic Functions

- heuristic_uninformed(board, side):
- - Counts pieces: +1 normal, +2 king.
- heuristic_informed(board, side):
- - +1 per piece, +2 per king.
- - Bonus for central control.
- - Bonus for mobility advantage.

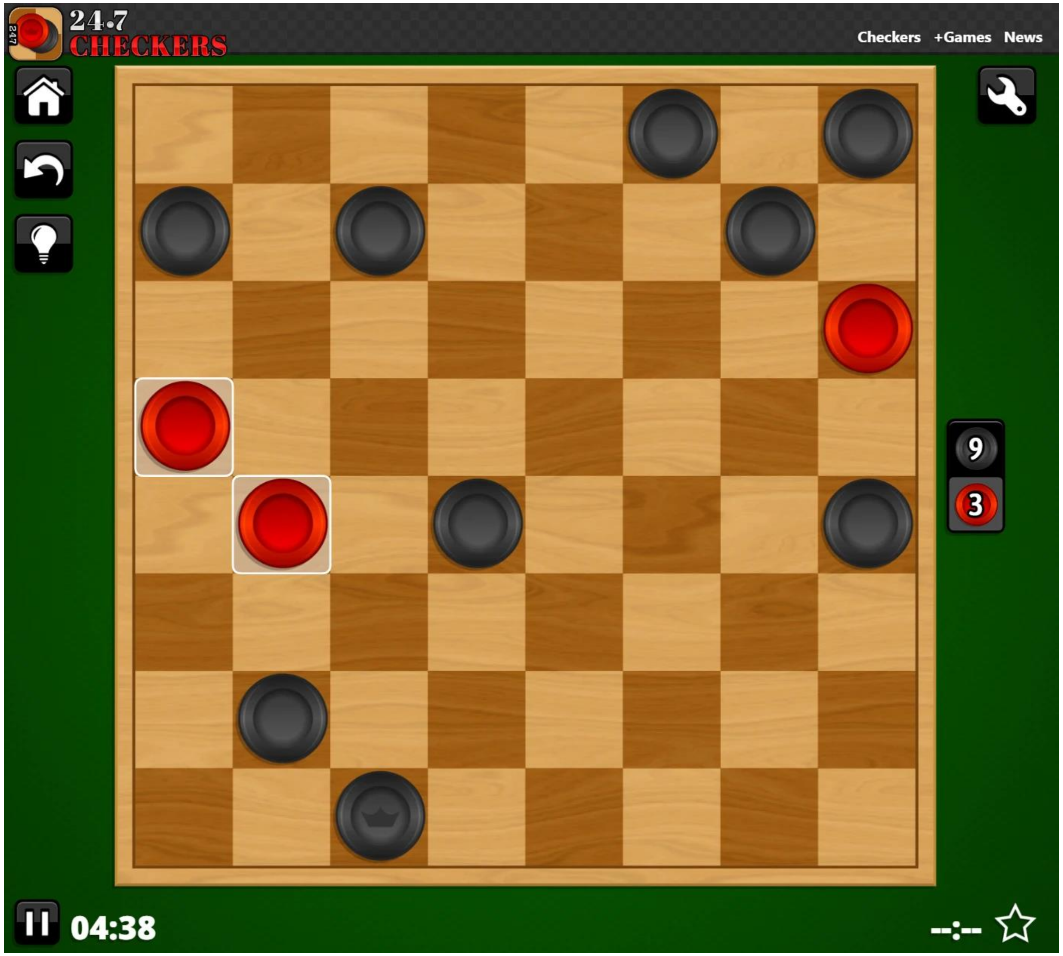
Code: Minimax (Alpha-Beta)

```
def minimax(board, depth, alpha, beta, maximizing, side):
    if depth == 0 or game_over(board):
        return heuristic(board, side), None
    if maximizing:
        maxEval = -inf; best_move = None
        for move in get_moves(board, side):
            new_board = deepcopy(board)
            apply_move(new_board, move)
            eval, _ = minimax(new_board, depth-1, alpha, beta, False, opp)
            if eval > maxEval:
                maxEval = eval; best_move = move
            alpha = max(alpha, eval)
            if beta <= alpha: break
        return maxEval, best_move
    else:
        minEval = inf; best_move = None
        for move in get_moves(board, side):
            new_board = deepcopy(board)
            apply_move(new_board, move)
            eval, _ = minimax(new_board, depth-1, alpha, beta, True, opp)
            if eval < minEval:
                minEval = eval; best_move = move
            beta = min(beta, eval)
            if beta <= alpha: break
        return minEval, best_move
```

Game Tree Analysis

- Minimax tree:
- \- Nodes = board states
- \- Max nodes = AI turn
- \- Min nodes = Player turn
- Depth 4: ~b^4 nodes (b = branching factor).
- Alpha-Beta pruning reduces explored nodes.
- BFS explores all nodes level-by-level, no pruning.

Example scenario



Advantages of alpha-beta pruning

- If branching factor **b = 4**:
  - Nodes without pruning: 341
  - Best-case with alpha–beta: 21
  - Nodes saved ≈ 341 − 21 = 320 → **~93.8%** pruned.

- **Faster decision-making**
- Reduces number of nodes evaluated compared to plain Minimax.

- **Same optimal result**
- Guarantees the **same move** as Minimax (if evaluation function is same), just faster.

- **Deeper lookahead**
- Time saved can be used to search **deeper** within the same time budget.

- **No extra memory cost**
- Uses **same O(depth)** memory as depth-first Minimax.

- **Improves with move ordering**
- Sorting moves by heuristic priority gives **maximum pruning benefit**.

Learnings

•**Understood Minimax Algorithm**
•Learned how AI evaluates moves using alternating **maximizing and minimizing** layers.
•**Applied Alpha–Beta Pruning**
•Experienced how pruning reduces search space without affecting the final result.
•**Explored Uninformed Search (BFS)**
•Implemented BFS to explore possible moves without heuristics.
•**Explored Informed Search (Heuristic-based)**
•Applied custom heuristic evaluation to guide AI's search for better efficiency.
•**Analyzed Game Trees**
•Built and interpreted game trees up to a given depth.
•**Evaluated Heuristics**
•Compared **simple** (piece count) vs **advanced** (position, kings, mobility) heuristics.
•**Observed Impact of Depth**
•Learned how increasing depth changes **AI's foresight** and computation time.
•**Experienced Game Logic**
•Understood rules of checkers including **movement, capturing, and king promotion**.
•**Practical AI Integration**
•Combined game mechanics with AI algorithms into a playable application.