Artificial Intelligence

BS (CS) _SP_2025

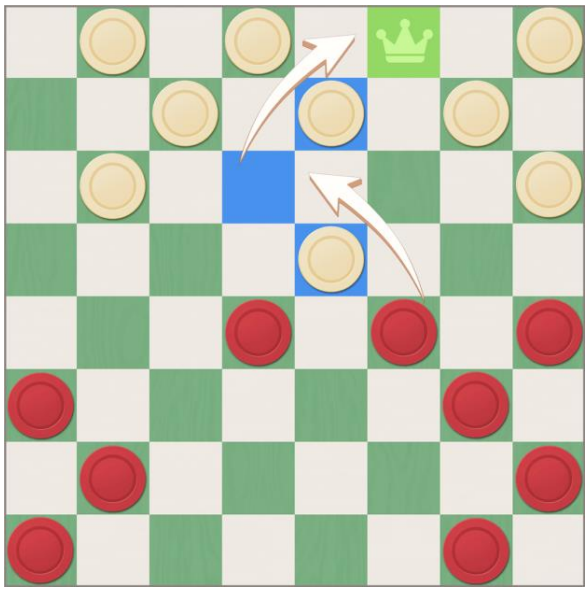
Lab_10 Tasks

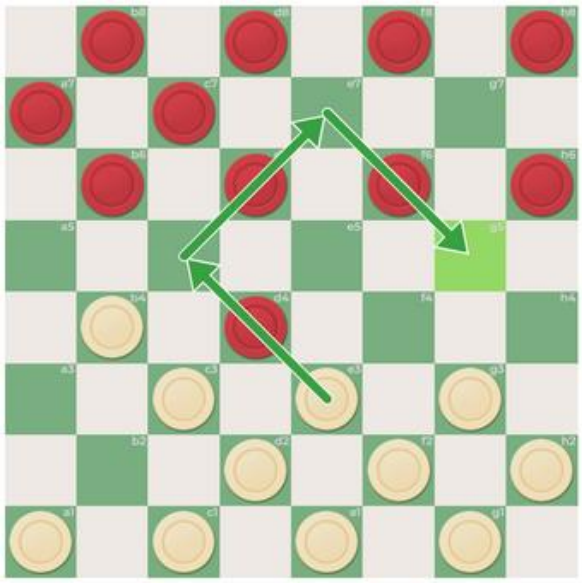
Learning Objectives:

1. Adversarial Searches (Alpha Beta Pruning)

Lab Task: Implement Checkers AI with Alpha-Beta Pruning

Checkers is a two-player strategy board game played on an 8x8 grid. Each player begins with 12 pieces placed on alternating dark squares across three rows. Players move diagonally and attempt to capture opponent pieces by jumping over them. Pieces that reach the opposite side of the board are promoted to kings, which can move both forward and backward diagonally.
The game ends when a player has no legal moves left.





Rules Summary:

1. **Initial Setup:**
   - The game is played on an 8×8 board using only the dark squares (where row + column is even).
   - Each player starts with 12 pieces placed on the first three rows of their side.
   - Red ('r') starts at the top, Black ('b') at the bottom.
   - The middle two rows are initially empty.
   - Light squares are not used for moves or piece placement
2. **Movement:**
   - Regular pieces move diagonally forward by one square.
   - Kings (promoted pieces) move diagonally in any direction.
3. **Captures:**
   - If a player's piece can jump over an adjacent opponent's piece into an empty square, it must do so.
   - Only single jumps are implemented in this lab (multi-jumps optional).
4. **Promotion:**

- A regular piece becomes a king ('R' or 'B') when it reaches the farthest row (opponent's home row).

5. **Winning Condition:**
   - A player wins if they eliminate all opponent pieces or leave the opponent with no legal moves.

Implementation Steps:

Step 1: Board & Game Logic

- Use an 8x8 array to represent the board.
- Use symbols like 'r', 'R' for Red pieces and kings; 'b', 'B' for Black.
- Implement move validation, captures, and promotion logic.
- Display the board in a clear format.

Step 2: Alpha-Beta Pruning

- Implement Minimax with Alpha-Beta pruning.
- Alternate between maximizing (AI) and minimizing (opponent).
- Prune subtrees when alpha ≥ beta.
- Use depth-limited recursion to simulate future moves.

Step 3: Evaluation Function

Design a heuristic to evaluate a board state. A basic function can be:
score = (#Regular pieces × 1) + (#Kings × 2)

Step 4: CLI User Interface

- Print the board and current score.
- Prompt the user to select from a list of valid moves.
- Allow human vs AI or AI vs AI play.

Skeleton Code:

```
import sys
import copy

BOARD_SIZE = 8

class Checkers:
    def __init__(self):
        """
        Initialize the checkers board and set the current player.
        Use 'r' and 'R' for red pieces and kings, 'b' and 'B' for black.
        Place 12 red and 12 black pieces on dark squares.
        """
        pass

    def create_board(self):
        """
        Set up the initial 8x8 board:
        - Red pieces on top 3 rows, black pieces on bottom 3 rows.
        - Only place pieces on dark squares ((row + col) % 2 == 0).
        """
        pass

    def print_board(self):
        """
        Display the current board with row and column numbers.
        """
        pass

    def get_all_valid_moves(self, player):
        """
        Return a list of all valid moves for the given player ('r' or 'b').
        Include moves for regular and king pieces, and prioritize captures.
        """
        pass

    def get_valid_moves_for_piece(self, row, col):
        """
        Given a piece's position, return all valid move options for that piece.
        Account for direction (forward for regular, both for king) and captures.
        """
        pass

    def make_move(self, move, player):
```

```
        """
        Apply the move to the board:
        - Move the piece
        - Remove any captured opponent piece
        - Promote to king if piece reaches the last row
        """
        pass

    def is_game_over(self):
        """
        Check whether the game is over (no moves or no pieces for a player).
        """
        pass

    def evaluate(self):
        """
        Calculate board score: (1 × regular pieces) + (2 × kings).
        Return (AI's score - Opponent's score) to guide AI decisions.
        """
        pass

    def alphabeta(self, depth, alpha, beta, maximizing_player):
        """
        Use Minimax with Alpha-Beta Pruning to select the best move.
        """
        pass

def play_human_vs_ai():
    """
    Launch a human vs AI game of Checkers in the console.
    AI (Black) uses Alpha-Beta Pruning; human plays as Red.
    Display board, show legal moves, and accept input for human turns.
    """
    pass

# Entry point
if __name__ == "__main__":
    play_human_vs_ai()
```