# Checkers with minimax

Bekzhan Omirzak, omirzbek@fit.cvut.cz

## 1. Introduction

This project explores the construction of a perfect strategy for a simplified variant of the classic board game Checkers on a 4×4 board, in alignment with the task "Neomylný šachista". In contrast to full-sized games, small board configurations are more suitable for exhaustive analysis and strategic exploration due to their limited state space. The reduced scale enables the use of game-solving algorithms to approach optimal (or perfect) play. Our implementation simulates a human vs. AI game where the AI always plays optimally using a minimax algorithm with alpha-beta pruning, a standard method in adversarial game search.

## 2. Approach

### Game Representation

The game is modeled using object-oriented programming in Python. The main components are:

- Board: Manages the game state, including the 4×4 grid, piece positions, turn alternation, move validation, capturing, promotion logic, and win/draw conditions. It also provides utility methods for evaluating the board and generating all legal moves for a given player.
- Man, and King: Represent the two types of checkers pieces. These classes provide legal move generation and movement logic.

### Handling Game Over

The system detects three end states:

- Win: When one player has no pieces left.
- Draw: When neither player can move or 20 moves have passed without a capture/promotion.

### User Interface (UI)

The UI provides a simple drag-and-drop system for human moves:

- Only black (human) pieces are draggable.
- Illegal moves snap the piece back.
- After each move, the AI responds automatically after a short delay.
- Popups are used for game start, end, and restart options.

### AI Strategy: Minimax

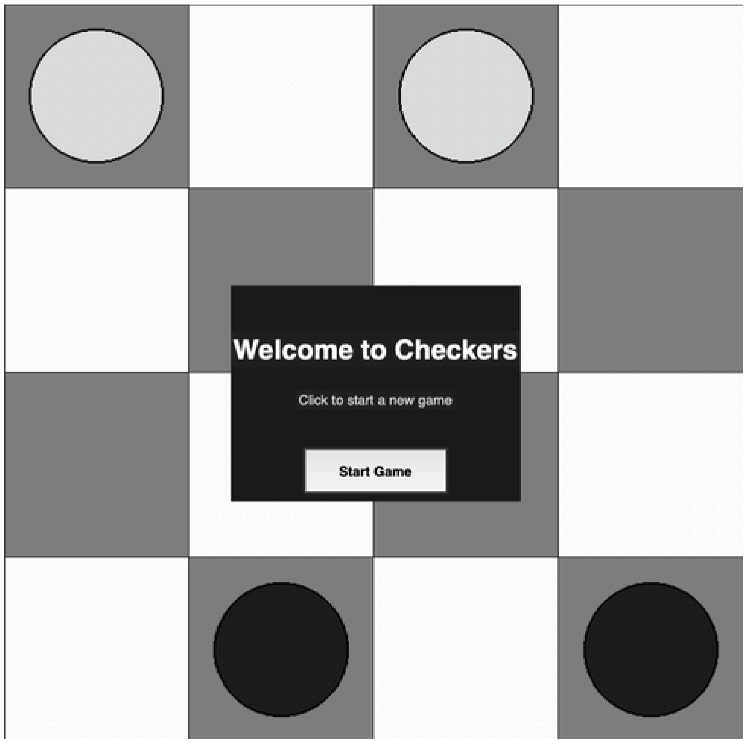
To find the best move for the AI, the program uses the Minimax algorithm, enhanced with alpha-beta pruning to reduce the number of states evaluated. The algorithm recursively simulates all possible moves up to a fixed depth. For each move, it uses the **get_all_moves()** method to gather all legal moves for the current color. Each resulting board state is evaluated using **evaluate_board()**, which assigns scores based on the number and type (man or king) of pieces on the board.

## 3.Implementation

**Backend**:Python

**Frontend:** Tkinter(Python)

## 4.Examples



Starting menu and board in the background

## 5. Conclusion

This project successfully demonstrates the implementation of a simplified checkers game on a small 4×4 board with an AI capable of making optimal decisions using the Minimax algorithm with alpha-beta pruning. The combination of a visual frontend using Tkinter and a well-structured backend allowed for clear separation of logic and presentation. Although the reduced board size simplifies the problem space, the core principles of game AI—such as move generation, evaluation, and strategic planning—are preserved.

## 6. References

- Minimax
- Tkinter